Implementación HW de Arquitectura de Medida de Distancias por IR con Mitigación de Multicamino

Máster Universitario en Sistemas Electrónicos Avanzados.
Sistemas Inteligentes
Departamento de Electrónica

Presentado por:
D. Adrián Ruano González

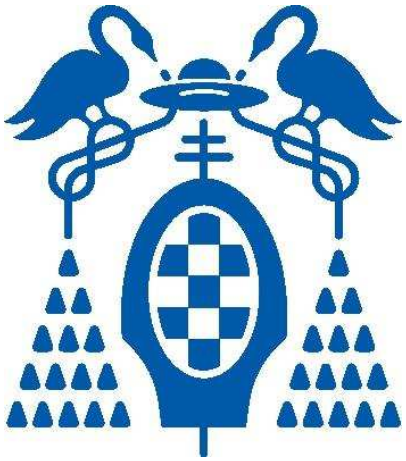
Dirigido por:
D. Ernesto Martín Gorostiza y D. David Salido Monzú

Alcalá de Henares, a 22 de Septiembre de 2014

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster Universitario en Sistemas Electrónicos Avanzados.
Sistemas Inteligentes



Trabajo Fin de Máster

Implementación HW de Arquitectura de Medida de Distancias por IR con Mitigación de Multicamino

Adrián Ruano González

2014

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Sistemas Electrónicos Avanzados.
Sistemas Inteligentes

Trabajo Fin de Máster

Implementación HW de Arquitectura de Medida de Distancias por IR con Mitigación de Multicamino

Autor: Adrián Ruano González

Directores: Ernesto Martín Gorostiza, David Salido Monzú

Calificación: .....

Fecha: .....

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mi familia por el inmenso y constante apoyo que me han dado a lo largo de estos dos años, porque no hay nada más satisfactorio que ver vuestra alegría a medida que he ido avanzando en este arduo viaje. Este trabajo supone el broche final a mi etapa universitaria, una etapa llena de buenos momentos, y otros no tan buenos. Y es en esos momentos cuando vosotros siempre habéis estado ahí, cuando más lo he necesitado para ayudarme en la medida de lo posible de un modo u otro y animarme para poder superarlos. Nunca habéis parado de creer en mí y de recordarme lo perfectamente capaz que soy de todo esto y mucho más, lo cual en muchas ocasiones yo mismo he olvidado. Muchas gracias por darme fuerzas en esos momentos de flaqueza, que no han sido pocos, ya que sin vosotros jamás habría logrado llegar hasta donde estoy hoy.

También quiero dar las gracias a mis directores de este trabajo, Ernesto Martín Gorostiza y David Salido Monzú, a quienes les debo la oportunidad de haber podido realizar este proyecto, porque sin ellos no hubiese sido posible. Para mí ha sido un trabajo bonito y muy interesante, que me ha dado la posibilidad de poder aprender y hacerme crecer aún más. Muchas gracias por vuestra dedicación, paciencia, y esfuerzo conmigo día a día.

Por último, darle las gracias a Álvaro Hernández Alonso por su ayuda y sus consejos, que han sido vitales para que este trabajo llegue a buen puerto, y agradecerle también la gran disposición que ha mostrado hacia mí.

Quizá estas palabras se queden cortas o yo no lo sepa expresar mejor, pero es todo cuanto puedo deciros, y sabed de verdad que mi agradecimiento es infinito y os deseo lo mejor.

Muchísimas gracias a todos, Adrián.

Resumen

En algunas de las tecnologías empleadas para implementar sistemas de posicionamiento local las interferencias por multicamino constituyen actualmente una de las principales fuentes de error. Por ello, en este trabajo se propone la implementación hardware de una novedosa arquitectura de medida diferencial de distancias por IR con mitigación del multicamino. Esta arquitectura está destinada a un sistema de posicionamiento local enfocado hacia la robótica móvil dentro de espacios inteligentes.

Teniendo en cuenta los requerimientos funcionales del sistema de medida se han analizado distintos dispositivos hardware con el objetivo de encontrar el más idóneo en cuanto a rendimiento y coste. Finalmente, la implementación de la arquitectura de medida ha sido llevada a cabo sobre una FPGA mediante el lenguaje de descripción hardware VHDL. Una vez implementada la arquitectura, ésta ha sido sometida a diversas pruebas experimentales para verificar su correcta funcionalidad en condiciones no ideales (dinámica del robot móvil, ruido, e interferencias por multicamino), mostrando un gran rendimiento ante las mismas.

Palabras clave: LPS, mitigación del multicamino, medida óptica diferencial de distancia, FPGA.

Abstract

Nowadays multipath interferences constitute one of the main error sources in some of the technologies used to implement local positioning systems. Hence, in this master thesis a hardware implementation of a novel IR differential ranging architecture with multipath mitigation is proposed. This architecture is designed for a local positioning system focused on mobile robotics inside intelligent spaces.

Taking into account the functional requirements of the ranging system, different possible hardware devices have been analyzed in order to find the most suitable regarding performance and cost. Finally, the ranging architecture implementation has been carried out on a FPGA device by hardware description language VHDL. Once the architecture has been implemented, several experimental tests to verify its proper functionality in non ideal conditions (mobile robot dynamic, noise, and multipath interferences) have been carried out, showing a great performance in those conditions.

Keywords: LPS, multipath mitigation, differencial distance optical measurement, FPGA.

Índice general

Índice de figuras

Índice de tablas

Códigos fuente

Capítulo 1

Introducción

1.1 Introducción

En la actualidad, los LPS's *(Local Positioning Systems)* constituyen un área de investigación y desarrollo emergente en el campo del posicionamiento aplicado a robótica móvil y transporte inteligente en espacios interiores. Los LPS's han surgido como alternativa a los GNSS's *(Global Navigation Satellite Systems)* para suplir las carencias de éstos en espacios interiores. Las limitaciones que presenta un GNSS son: por un lado, su precisión (metros), salvo en GNSS's de precisión (más caros, complejos y lentos, y no presentan una gran mejora), es válida para el posicionamiento de vehículos en carreteras, pero resulta inaceptable para el posicionamiento de un robot móvil en un espacio de dimensiones reducidas donde se puede llegar a requerir una precisión de centímetros; por otro lado, funciona mediante señales de radiofrecuencia, las cuales sufren efectos de atenuación y reflexión al atravesar materiales sólidos, lo que empeora aún más la precisión en espacios interiores.

Una de las aplicaciones con mayor auge en los últimos años a la que se destina el uso de los LPS's son los EI's *(Espacios Inteligentes)*. Un EI es un entorno capaz de captar información sobre lo que sucede en su interior, y en base a ella, tomar decisiones y llevar a cabo tareas concernientes a los elementos presentes en él para prestar algún tipo de servicio. Un EI está compuesto por: un potente sistema multisensorial, un sistema de procesado (central o distribuido) que recibe la información y decide, una infraestructura para localización (emisores, receptores, puntos de referencia, y balizas), una infraestructura de canales de comunicación, y actuadores, como pueden ser robots móviles que desempeñen diversas funciones. La interacción de los robots móviles debe ser segura tanto con personas como con elementos propios del EI, por lo que, de las numerosas tareas a las que el EI debe enfrentarse, conocer la posición de los robots móviles en tiempo real y con buena precisión es clave. Es por este motivo por el cual los LPS's se encuentran en un continuo proceso de evolución. Bajo esta misma filosofía de espacios inteligentes existen entornos definidos como: *Ambient Inteligent*, *Context-Aware Enviroment*, *Pervasive Computing*, etc.

Las tecnologías en las que se basan los LPS's son de diversa naturaleza. Hasta el momento, las cámaras y ultrasonidos han sido las tecnologías más desarrolladas. Hoy en día se está trabajando en infraestructuras de radiofrecuencia *wireless* como UWB *(Ultra Wide Band)*, ó RFID *(Radio Frequency Identification)*. Como alternativa novedosa, el LPS propuesto en este trabajo se basa en el NIR *(Near InfraRed)*.

La posición del robot móvil estimada por el LPS puede ser obtenida por medida de ángulos (triangulación), o por medida de distancias (trilateración). En trilateración, cualquier distancia, o diferencia de

distancias, puede obtenerse por la medida del tiempo de vuelo (TOF[1]), que es el tiempo que transcurre desde que una señal es emitida hasta que es recibida; o diferencia de tiempos de vuelo (DTOF's[2]). La medida mediante TOF's ó DTOF's es habitual cuando se emplean ultrasonidos, sin embargo, en el caso del IR es necesaria una precisión de picosegundos para lograr precisiones en distancia de centímetros, y actualmente es extremadamente complicado conseguir un hardware y canales ópticos adecuados con esa resolución temporal. Otros sistemas trabajan mediante RSSI *(Received Signal Strength Indicator)*, un método que consiste en la medida del nivel de señal recibida, y que ha sido descartado debido a su baja precisión. En el sistema propuesto se utiliza un método de posicionamiento basado en el cálculo de medidas diferenciales de distancia entre el robot móvil y parejas de puntos de referencia. Estas diferencias se obtienen a partir de la estimación de la medida del desfase sufrido por la señal IR emitida por el robot móvil hasta alcanzar un receptor situado en cada punto de referencia. La medida del desfase resulta una interesante alternativa respecto a las medidas realizadas mediante TOF's y DTOF's.

La problemática en el desarrollo del LPS IR puede sintetizarse en torno a dos aspectos: las interferencias por multicamino, y las propias características del LPS. Las interferencias por multicamino suponen un importante problema en varias de las tecnologías empleadas para implementar LPS's. Este tipo de interferencias son causadas por todas las contribuciones de señal que llegan indirectamente (reflexiones y difracciones) desde un emisor a un receptor, empeorando la precisión en la estimación de la posición. La problemática asociada a las características del propio LPS radica en conseguir una alta precisión en la estimación de distancia y al mismo tiempo una amplia cobertura y velocidad admisible del robot móvil, debido al compromiso existente entre el SNR *(Signal-to-Noise Ratio)* y la direccionalidad ancho de banda de los dispositivos emisores y receptores [1].

1.2 Motivación y objetivos

La implementación del LPS IR en tiempo real representa un importante reto tecnológico dadas las elevadas frecuencias de trabajo y digitalización exigidas para garantizar la mitigación de las interferencias por multicamino, frecuencias cercanas al límite tecnológico actual de los sistemas electrónicos empotrados.

El objetivo de este trabajo es definir el diseño e implementación hardware de una arquitectura de medida diferencial de distancias con mitigación de las interferencias por multicamino previamente definida, y comprobar si es capaz de mejorar la precisión en la medida de distancias, y por tanto, la precisión en la estimación de la posición del robot móvil.

1.3 Organización de la memoria

La memoria de este trabajo se organiza en cuatro grandes capítulos. En el primero de ellos, el contexto, se realiza un análisis de los estados actuales del LPS IR y de los métodos y técnicas de mitigación del multicamino; a continuación se describe el sistema de medida diferencial de distancias en el que se basa este trabajo y que intenta resolver la problemática del multicamino. El segundo capítulo consiste en una propuesta de implementación hardware del sistema descrito anteriormente: cuáles son los requisitos que exige la arquitectura, alternativas de soportes de implementación, e implementación del sistema en sí. En el tercer capítulo se muestran los resultados obtenidos tras someter a la arquitectura de mitigación a distintas pruebas que permitan validarla. El cuarto y último capítulo contiene las conclusiones que se han extraído de este trabajo, y se plantean trabajos futuros que puedan mejorar el actual.

[1] TOF: Time of Flight.
[2] DTOF: Differential Time of Flight.

Capítulo 2

Contexto

2.1 Background

Como ya se explicó en la sección 1.1, uno de los principales problemas en el desarrollo de LPS's son las interferencias por multicamino. De aquí surge la necesidad de definir un nuevo sistema de medida que proporcione una solución eficiente a este problema. Dicho sistema se describe en el apartado 2.3.2, y posteriormente es implementado a lo largo del capítulo 3, siendo esta tarea el principal objetivo de este trabajo. Esta nueva arquitectura debe poder ser aplicable sobre el LPS IR digital desarrollado previamente en el Departamento de Electrónica de la Universidad de Alcalá. Se ha demostrado el buen rendimiento del sistema de medida dado que los errores de posicionamiento obtenidos están por debajo de 2.5 cm.

El LPS IR digital en el que se basa este trabajo está compuesto por un subsistema de medida y un subsistema de estimación. El subsistema de medida establece las relaciones métricas entre el objeto a posicionar y diversos puntos del entorno de posición conocida. El subsistema de estimación estima matemáticamente la posición del objeto dentro del sistema de referencia a partir de las diferencias de distancia proporcionadas.

El subsistema de estimación realiza el posicionamiento del robot móvil mediante un proceso de trilateración hiperbólica y estimación por mínimos cuadrados no lineales a partir de, al menos, tres medidas diferenciales de distancia entre el robot móvil y parejas de puntos de referencia situados en el plano superior del entorno; como la altura del emisor IR es conocida, basta con sólo dos medidas. Las diferencias de distancias son obtenidas a partir de diferencias del desfase que experimenta la señal IR emitida por el robot móvil hasta alcanzar los receptores situados en los puntos de referencia. La conversión fase-distancia se realiza aplicando las ecuaciones (2.1) y (2.2), que establecen la relación entre la distancia recorrida d por camino directo de la señal y el desfase medido φ en función de la longitud de onda de la señal λ_m.

$$K_{d/\varphi} = \frac{d}{\varphi} = \frac{\lambda_m}{360^\circ} \quad [m/^\circ] \tag{2.1}$$

$$\lambda_m = \frac{c}{f_m} \quad [m] \tag{2.2}$$

El robot móvil que se desplaza dentro del entorno lleva a bordo un emisor IR de 940 nm que emite una señal modulada en intensidad por un tono de 8 MHz. Para $c = 3 \cdot 10^8$ m/s y $f_m = 8$ MHz se tiene una longitud de onda $\lambda_m = 37.5$ m, estableciéndose una relación $K_{d/\varphi}$ de 10.41 cm/°: $d = 0.1041 \cdot \varphi$ [m]. La elección de la frecuencia de modulación no es sencilla, pues resulta del compromiso existente entre el

correcto funcionamiento de los dispositivos fotónicos del enlace óptico y la adecuada sensibilidad $K_{d/\varphi}$. Es fundamental minimizar la incertidumbre en la medida de fase, pues una desviación de tan sólo 0.5º en la medida de fase introduce un error de aproximadamente 5 cm en la medida de distancia.

En la Figura 2.1 se representa el diagrama de bloques del sistema de posicionamiento completo: en azul se muestra el sistema sensorial, en amarillo el subsistema de medida diferencial de distancia, y en verde el subsistema de estimación de la posición. La señal emitida por el robot móvil debe tener la máxima potencia posible en un ángulo sólido bastante amplio para que llegue con potencia suficiente a todos los receptores del entorno. El entorno definido para las pruebas, denominado CBL *(Celda Básica de Localización)*, tiene unas dimensiones de 3x3x3 m, y dispone de cinco receptores en el plano superior, cada uno de los cuales envía la señal que recibe a una tarjeta digitalizadora tras una etapa previa de acondicionamiento. El receptor que presenta mayor potencia de señal en cada momento es el que actúa como referencia.

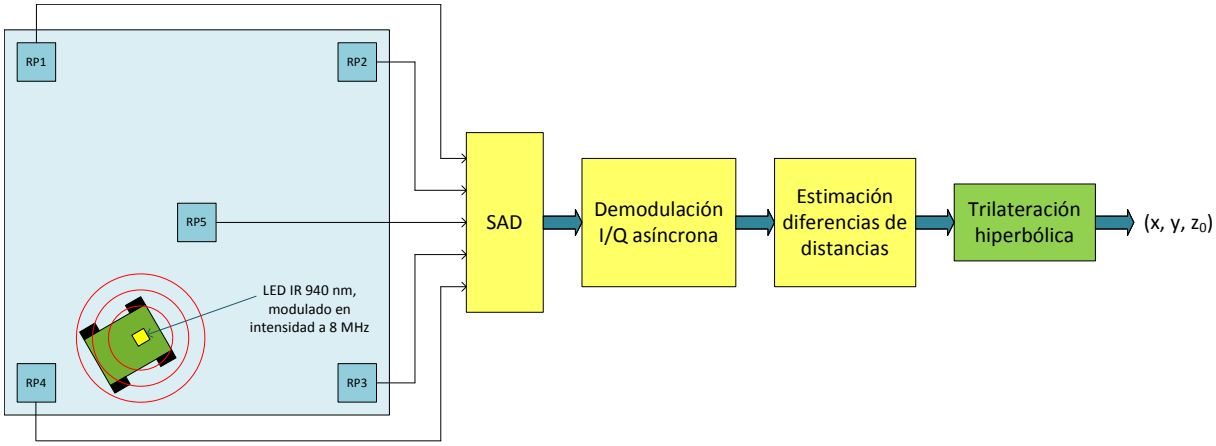


Figura 2.1: Diagrama de bloques del LPS IR digital.

El subsistema de medida realiza sobre cada una de las señales digitalizadas un proceso de demodulación I/Q *(In-phase/Quadrature)* asíncrono que permite obtener la fase diferencial ellas. Las diferencias de fase son calculadas respecto a la fase de la señal que actúa como referencia, y a continuación se realiza un mediado para mejorar el SNR de las medidas diferenciales. Las medidas de fase diferenciales son finalmente convertidas a medidas de distancia diferenciales aplicando las ecuaciones (2.1) y (2.2) de forma que puedan ser empleadas por el subsistema de estimación para la obtención de la posición del robot móvil. En la Figura 2.2 se muestra un diagrama de bloques del subsistema de medida.

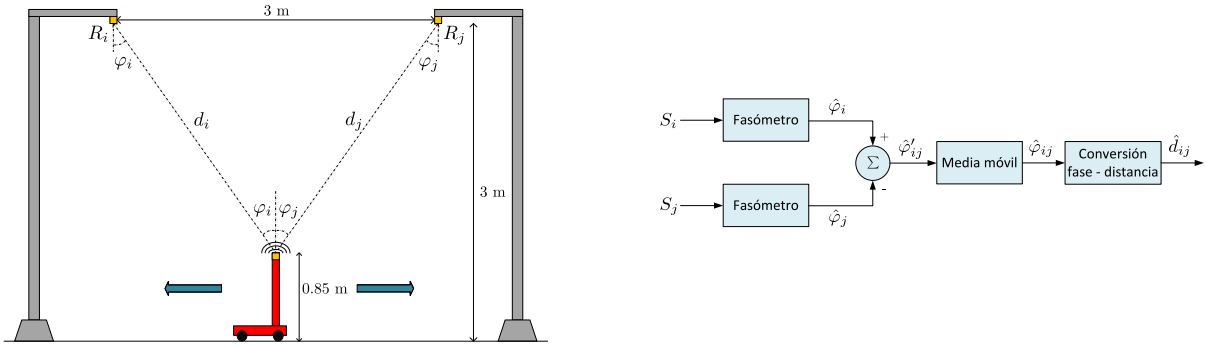


Figura 2.2: Subsistema de medida diferencial de distancia.

2.2 Estado del Arte

Las interferencias por multicamino son un importante problema en varias de las tecnologías de implementación de LPS. Dentro del ámbito de los GNSS's es donde más han sido estudiadas y más técnicas de mitigación se han desarrollado; en menor medida también han sido objeto de estudio en UWB e interferometría.

2.2.1 Mitigación del multicamino en GNSS's

A pesar de las mejoras introducidas en la tecnología de los GNSS's durante los últimos años, el problema del multicamino continúa siendo un desafío a superar en aplicaciones que requieren alta precisión. El problema del multicamino radica en la elevada correlación que existe entre una réplica de la señal directa retarda y atenuada, y la propia señal directa. Debido a ello, se hace complicado distinguir todas las componentes de multicamino de la componente directa y eliminarlas de la señal, sobre todo cuando el retardo entre las réplicas y la componente directa es pequeño. Para solucionar el problema del multicamino existen actualmente dos perspectivas principales: clásica y moderna.

A lo largo de las últimas décadas se han propuesto e implementado diversas técnicas para mitigar los errores por multicamino a nivel de procesado de la señal recibida en banda base: NC *(Narrow Correlator)* [2], SEC *(Strobe & Edge Correlator)* [3, 4] , GC *(Gated Correlator)* [5, 6], y HRC *(High-Resolution Correlator)* [6]. Este tipo de técnicas, que se engloban dentro de la perspectiva clásica, consiguen mitigar el multicamino de la fase del código empleando señales de referencia especialmente diseñadas para ser correladas con la señal entrante contaminada por las componentes de multicamino. Estas técnicas tienen como ventaja la facilidad de implementación, y además son independientes del número de componentes de multicamino. Sin embargo, presentan varios inconvenientes que eclipsan a sus beneficios: bajo rendimiento en lo que a robustez frente al ruido se refiere en comparación con el procesado E-L estándar *(Early-Late)*, son menos efectivas en la mitigación debido a la gran pérdida del nivel de SNR, y además pueden proporcionar grandes niveles de offset si las componentes de multicamino tienen mayor potencia que la componente directa de la señal.

En las últimas décadas ha proliferado la perspectiva moderna y se ha puesto un mayor interés en otro tipo de técnicas conocidas como métodos de estimación del multicamino. Los métodos más destacados son los de estimación del multicamino por bucle enganchado en retardo (MEDLL[1]) [7] y sus variantes MMT *(Multipath Mitigation Technique)* [8], VC *(Vision Correlator)* [9], CADLL *(Coupled Amplitude Delay Lock Loop)* [10], y ECADLL *(Enhanced Coupled Amplitude Delay Lock Loop)* [11]. La principal ventaja de estos métodos es que son capaces de estimar la fase del código y de la portadora tanto de la componente directa como de las componentes multicamino de la señal, y además pueden llegar a alcanzar los límites del rendimiento teórico. Para lograr alcanzar el rendimiento adecuado estás técnicas requieren una elevada frecuencia de muestreo y el empleo de un mayor número de correladores; todo ello conlleva un incremento en la complejidad del sistema, en el consumo de potencia, y en los costes computacionales. Además, estos métodos de estimación también requieren un nivel de SNR muy alto, ya que son más sensibles al ruido, lo cual puede conseguirse con un mayor tiempo de integración. Sin embargo, estas técnicas se encuentran limitadas a aplicaciones estáticas. Por último, han aparecido otros métodos de estimación como la técnica de Teager-Kaiser (TK) y MUSIC *(MUltiple SIgnal Classification)*, métodos que a día de hoy se encuentran lejos de una posible implementación práctica o dependen de un *setup* más complejo.

[1] MEDLL: Multipath Estimation Delay Lock Loop.

A continuación se describen las técnicas clásicas de mitigación del multicamino mencionadas anteriormente:

- *Narrow Correlator*: históricamente los receptores GPS *(Global Positioning System)* convencionales han empleado correladores E-L con un espaciado de 1.0 chip de código C/A *(Coarse/Acquisition)* para implementar bucles enganchados en retardo (DLL[2]). El estrechamiento del espaciado de la correlación proporciona una gran ventaja, que es la reducción de errores en el seguimiento de código C/A en presencia de ruido y del multicamino. La reducción del ruido se consigue gracias a que las componentes de ruido de las señales E-L son correladas, y por tanto, tienden a anularse (siempre y cuando las señales E-L sean simultáneas). Los efectos del multicamino se reducen debido a que el discriminador DLL está menos distorsionado por las componentes de multicamino. Sin embargo, el estrechamiento del espaciado también conlleva desventajas: requiere un ancho de banda de pre-correlación más amplio, y frecuencias de muestreo y procesado digital más elevadas.

  En [12] se presenta el NovAtel GPSCardtm, un receptor GPS con el espaciado de la correlación variable. Este receptor es capaz de variar el espaciado de los códigos de referencia E-L de 0.05-1.0 chip de código C/A. La principal razón para implementar el estrechamiento del espaciado es que se consigue un rendimiento superior en el seguimiento de código C/A en presencia de ruido. La clave de esta técnica, ilustrada en la Figura 2.3, reside en la selección del registro de desplazamiento del reloj que genera los códigos *early*, *punctual*, y *late*. El parámetro N puede tomar valores desde 1 hasta 20 para proporcionar un espaciado variable entre códigos; de esta forma se tiene un código E-L espaciado de 0.05-1.0 chip de código C/A. Sin embargo, cuando el espaciado del correlador es muy estrecho el rendimiento del DLL se ve fuertemente afectado por el ancho de banda de pre-correlación. Ello se debe a que la banda restrictiva tiende a redondear el pico de auto-correlación, y por tanto, limita la discriminación entre las correlaciones *early* y *late*.

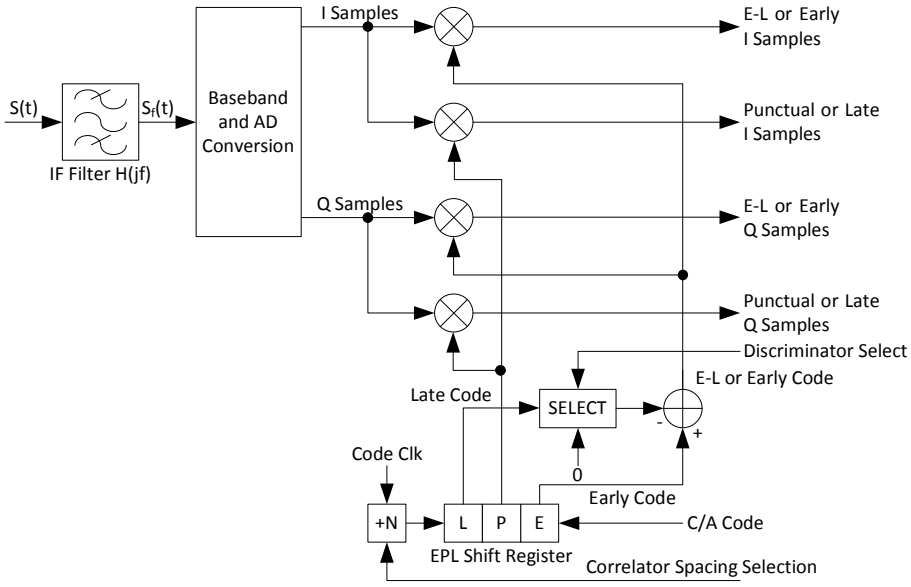


Figura 2.3: Correlador con estrechamiento del espaciado.

En cuanto a los efectos de multicamino, el espaciado estrecho consigue reducirlos debido a que en la función de correlación cruzada (R_x) la distorsión es menos acentuada en regiones próximas al pico que en otras regiones más alejadas del pico. Por tanto, si se sigue código C/A en una región próxima al pico de R_x los efectos de multicamino se reducen, siempre y cuando el DLL sea no coherente. El

[2]DLL: Delay Lock Loop.

estrechamiento del espaciado de la correlación en los DLL's no coherentes hace que el máximo error cometido debido al multicamino sea proporcional al espaciado, pero también condicionado por el ancho de banda de pre-correlación disponible. El error se reduce en la medida que se estrecha el espaciado y se incrementa el ancho de banda de pre-correlación, reduciéndose el radio a los objetos que causan el multicamino.

- *Strobe & Edge Correlator*: el correlador *strobe* es un acumulador del producto de una entrada que mezcla una señal de satélite con una señal interferente y con una señal de referencia denominada secuencia *strobe*. Un *strobe* consiste en varios pulsos de distinta longitud y amplitud, caracterizándose por su duración total y su perfil. Una señal de referencia en forma de secuencia *strobe* está enganchada en fase con la señal PRN *(PseudoRandom Noise)* generada en el receptor. El enganche con la señal PRN puede realizarse cuando ésta cambia de signo, o bien en un flanco de chip de la misma. El perfil del *strobe* tiene gran importancia, pues puede ser simple en forma de un único pulso, o complejo en forma de varios pulsos. Independientemente del perfil del *strobe*, siempre que un chip de la señal PRN cambia de polaridad, el *strobe* se invierte. Los correladores *strobe* generan la componente dI para el DLL coherente, mientras que las componentes I/Q son generadas por correladores convencionales.

  El perfil del *strobe* realmente controla las principales características del discriminador DLL (precisión, fiabilidad, ...); de esta forma, variando el perfil del *strobe* se varían las características del DLL para que se adapte a las condiciones externas. Cuando el perfil del *strobe* es simple, la caracterítica del discriminador DLL presenta un único punto de equilibrio estable en el sistema de seguimiento, mientras que cuando el perfil es complejo, aparecen varios cruces por cero que pueden ser puntos de equilibrio. Es deseable que en el punto de equilibrio estable la ganancia del discriminador así como el valor esperado de la componente I sean aproximadamente máximos. Para hacer que el DLL funcione en el punto de equilibrio deseado en el modo seguimiento, es necesario que el sistema de adquisición inicial proporcione un desfase inicial dentro del rango de adquisición y que el signo de la ganancia del DLL sea correcto.

  Los resultados mostrados en [4] indican que el correlador *strobe* es una herramienta bastante flexible para la adaptación del DLL a las condiciones externas mediante la elección de un perfil del *strobe* adecuado en un correlador de la señal de referencia. De este modo, un perfil de *strobe* simple enganchado con la señal de referencia PRN cuando ésta cambia de signo posibilita minimizar el ruido en el error de seguimiento. En cambio, un perfil de *strobe* complejo da lugar a un correlador *strobe* con baja sensibilidad a las señales reflejadas, ya que reduce el espacio donde pueden aparecer reflectores perjudiciales, y así mitigar el error de multicamino producido por éstos. Puesto que el error de multicamino controla la precisión real del DLL, los *strobes* de perfil complejo proporcionan grandes ventajas cuando se trabaja con receptores GPS basados en código C/A.

- *Gated Correlator*: en [13] se sugiere la posibilidad de mejorar el rendimiento frente al multicamino mediante el acotamiento las señales entrante y localmente generada para la correlación. Este concepto, denominado correlador acotado, se ilustra en la Figura 2.4 para señales acotadas centradas sobre las épocas de chip. Se asume que la anchura de puerta w es inferior a la mitad de la longitud de un chip. Hay dos casos a tener en cuenta: cuando se produce una transición de chip, o cuando el valor del chip permanece. Se asume además que la probabilidad de una transición de chip es de 0.5.

  El discriminador DLL se obtiene mediante correladores *Early-Late* acotados. La función de autocorrelación de un código C/A acotado es más estrecha respecto a la de un código P y C/A convencional, lo que se traduce en una mejora de la mitigación del multicamino tanto en la fase del

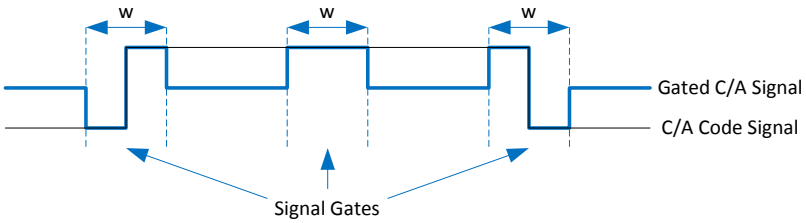


Figura 2.4: Concepto de correlador acotado.

código como de la portadora. Sin embargo, el pico de la función de auto-correlación acotada es muy inferior al obtenido con los códigos P y C/A, y por tanto, el rendimiento frente a ruido térmico se ve penalizado. El SNR de post-correlación sufre una pérdida proporcional al factor de acotamiento w. Para un elevado C/N_0 *(Carrier-to-Noise Ratio)*, el correlador acotado consigue un rendimiento frente al ruido en la medida del código equivalente al de un código C/A convencional con el mismo espaciado de correlación.

Hoy en día se presta especial atención al uso de modulaciones BOC *(Binary Offset Carrier)* en las señales de GNSS's. Las modulaciones BOC presentan funciones de auto-correlación con múltiples picos, lo que provoca que las técnicas de mitigación de multicaminos basadas en correlaciones (NC o HRC) sean menos óptimas. Por ello, en [14] se propone una técnica de mitigación del multicamino apropiada para señales BOC que permite eliminar los errores en las medidas de distancia en regiones de medio y largo retardo. Esta técnica recurre a funciones acotadas constituidas por pulsos *strobe* de tipo W2 ya que proporcionan un mejor rechazo al multicamino [4].

La estructura del receptor propuesta dispone de bucles de código y fase, y además incluye un mecanismo de detección y eliminación de falsos enganches del código. Para su implementación se necesitan tan sólo cuatro correladores (en comparación con los ocho que requiere HRC no coherente), y dos correladores extra para el mecanismo de de detección de falsos enganches simplificado. En esta estructura la señal de entrada es correlada con una señal BOC generada localmente, y con una señal acotada compuesta de pulsos *strobe* especialmente ajustados. Mediante el ajuste cuidadoso de la anchura del pulso *strobe* (L/T_c) y del ancho de banda del filtro de entrada (BT_c) es posible optimizar los siguientes aspectos: mitigación del multicamino, inmunidad al canal de ruido aditivo, y robustez a la pérdida del enganche del código.

La contribución de ruido térmico no puede ser despreciada ya que el rendimiento del receptor depende fuertemente de la cantidad de ruido del canal. La robustez del bucle de código frente al ruido térmico depende de la anchura del pulso *strobe*, ya que el bucle sólo es útil cuando el SNR supera un cierto umbral. A pesar de que el receptor propuesto tiene un rendimiento frente al ruido peor que el de los discriminadores no coherente E-L y coherente HRC, se compensa gracias a un rendimiento superior del discriminador de pulso *strobe* en presencia del multicamino.

Se ha demostrado que el error de multicamino decrece a medida que se incrementa el ancho de banda y la frecuencia de chip, y se reduce la anchura de los pulsos *strobe*. Debido al perfil asimétrico de la función de correlación cruzada se consigue un muy buen rendimiento en la mitigación de multicaminos de medio y largo retardo en comparación a un DLL E-L.

El receptor propuesto lleva consigo un compromiso entre la degradación causada por el ruido térmico y la capacidad de mitigación del multicamino. Por ello, su empleo está muy condicionado a las características del entorno, en concreto, dependerá de cuál sea la principal fuente de error.

- *High-Resolution Correlator*: esta técnica emplea la salida de múltiples correladores de un receptor GNSS convencional para obtener una aproximación a un GC. En [6] se asume que el receptor

dispone de cinco correladores espaciados entre ellos d chips: $Early_2$, $Early_1$, $Prompt$, $Late_1$, y $Late_2$ (E_2, E_1, P, L_1, y L_2). La idea de HRC es formar una combinación lineal a partir de la salida de los cinco correladores para obtener una función de correlación mucho más estrecha que la función de auto-correlación de un código C/A normal. El resultado de HRC es un rendimiento idéntico a GC para retardos del multicamino corto y medio, pero un error mayor cuando el retardo es de aproximadamente 1.0 chip. En la práctica, los retardos de multicamino normalmente son cortos o medios, por lo que GC y HRC presentan un rendimiento prácticamente equivalente. A pesar de que el rendimiento en la mitigación del multicamino de GC y HRC es similar (o incluso mejor) al obtenido en código P para retardos medios y largos, ambas técnicas son incapaces de mitigar multicaminos con retardos inferiores a 0.05 chip (los retardos más comunes en aplicaciones de medida y posicionamiento).

La técnica HRC no sólo mitiga el multicamino del código, sino que también proporciona una significativa mitigación del multicamino de la fase de la portadora comparada con las medidas de fase obtenidas con un correlador C/A convencional. Sin embargo, esta mejora conlleva una penalización en el rendimiento frente al ruido térmico respecto a otras técnicas. HRC sufre una degradación post-correlación del SNR proporcional al espaciado d en comparación con el seguimiento C/A convencional. Ello se debe a que el valor de pico de la función de auto-correlación de HRC es menor que en el caso de código C/A.

A continuación se describen las técnicas modernas de mitigación del multicamino mencionadas anteriormente:

- *Multipath Estimation Delay Lock Loop*: para reducir los errores de multicamino en el seguimiento en receptores DSSS *(Direct Sequence Spread Spectrum)*, en [7] se propone la optimización de las técnicas de estimación del receptor respecto al canal debilitado de multicamino. Esta nueva técnica se basa en estimaciones de máxima verosimilitud (ML[3]). A diferencia de los receptores convencionales de espectro ensanchado coherente en los que únicamente se correlaba la señal de entrada con el código de espectro ensanchado local, ahora deben restarse las funciones de correlación interferentes estimadas. Ello implica que en vez de calcular sólo el retardo y la fase de la señal directa, se calculen además el retardo y la fase de las señales de multicamino. La gran complicación de esta técnica reside en que el número M de señales de multicamino generalmente es desconocido y debe estimarse. La forma más simple de tratar con este problema es escoger un valor fijo de M que no sea muy pequeño, pues existirá un cierto error de multicamino, ni muy grande, ya que aumentará el error por ruido.

  La técnica de MEDLL puede ser implementada mediante dos alternativas. La primera de ellas, cuyo diagrama de bloques se muestra en la Figura 2.5, consiste en muestrear la señal de entrada durante T segundos y que el procesador calcule las ecuaciones que estiman el retardo y la fase de las señales directa y de multicamino. La correlación con la señal de entrada puede realizarse con un conjunto de correladores activos. Para reducir la carga computacional el procesador solamente usa una parte muestreada de la función de correlación. Los correladores sólo tienen que proporcionar R_{x0} (función de correlación cruzada de la señal directa), ya que el resto de términos R_{xi} se derivan fácilmente a partir del primero mediante la rotación del retardo y la fase. Una vez calculadas las estimaciones de retardo y fase de todas las señales, éstas son usadas para controlar los generadores de portadora y código en lazo cerrado.

  La segunda alternativa de implementación se basa en una estructura PLL *(Phase Lock Loop)* similar a un DLL coherente. En este caso son necesarios 2·(M+1) bucles para seguir las fases de todas las

[3]ML: Maximum Likelihood.

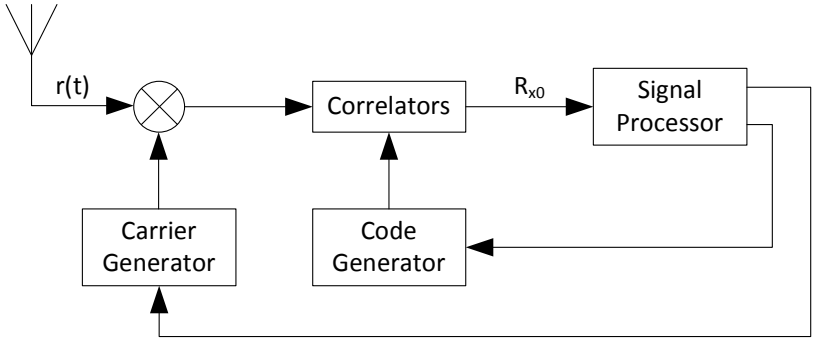


Figura 2.5: Diagrama de bloques de MEDLL.

portadoras y los retardos de todos los códigos, y M+1 correladores para estimar las amplitudes de todas las señales. La Figura 2.6 representa el diagrama de bloques de este tipo de implementación. Dispone de tres correladores para seguir los parámetros estimados (retardo, fase, y amplitud), los cuales son empleados para controlar los generadores de portadora y código. En la Figura 2.6 el parámetro d representa el espaciado de las correlaciones E-L. Para mejorar el rendimiento del DLL frente al multicamino se emplea un valor de d inferior a 1.0 chip [2].

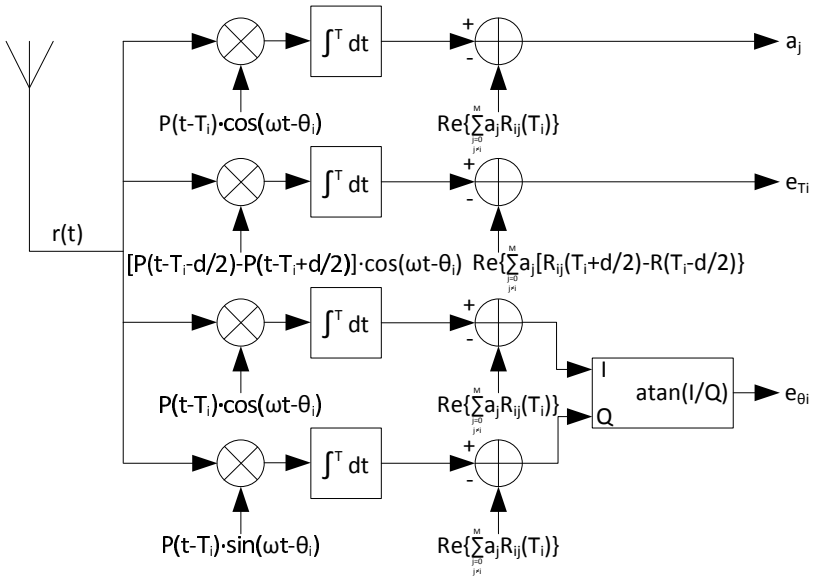


Figura 2.6: Diagrama de bloques de MEDLL basado en PLL.

La implementación de MEDLL puede simplificarse cuando la potencia de la señal directa es mayor que la potencia de las señales de multicamino, y no interesa reducir el error en la fase de la portadora. Esta versión simplificada de MEDLL es básicamente un DLL coherente convencional, con la diferencia de que la estimación del retardo del código está mejorada. Debido a ello, tiene la ventaja de eliminar los errores de multicamino en las estimaciones de retardo. Sin embargo, presenta los mismos inconvenientes que un DLL coherente convencional: pérdida de señal, y errores en la fase de la portadora. En las simulaciones realizadas en [7] se puede observar que el patrón típico de error de multicamino desaparece. Además, puede verse claramente que el rendimiento del MEDLL es muy superior al del DLL, rendimiento que podría mejorarse reduciendo aún más el espaciado de las correlaciones E-L [2].

La técnica de MEDLL consigue estimar el retardo y la fase de las señales a partir de un conjunto de ecuaciones de máxima verosimilitud. En ausencia de ruido estas estimaciones no presentan error

a diferencia de otros sistemas de estimación convencionales. Por tanto, MEDLL es una técnica que permite mantener una alta precisión en la medida dentro de entornos con propagación de multicamino. Aunque MEDLL ha sido desarrollada para receptores DSSS, puede extenderse fácilmente a otro tipo de sistemas.

- *Multipath Mitigation Technique*: es un método de estimación del multicamino derivado de MEDLL y basado en ML. Consiste en un algoritmo que procesa un array que contiene información sobre la forma de pulso para producir una estimación de la señal directa, y de una o varias señales de multicamino. Cada una de las señales se representa mediante tres parámetros: el retardo del código, y la amplitud y fase de la portadora. La gran ventaja de MMT es que permite reducir el número de parámetros de búsqueda en cada señal de tres a uno, el retardo del código, reduciendo así los costes computacionales en gran medida. Las fórmulas en las que se basa la técnica MMT dependen del número de señales de multicamino, número que generalmente es desconocido. La forma más común de proceder es emplear un banco de correladores para obtener un perfil de correlación cruzada (R_x) con gran detalle, y asumiendo que inicialmente no existe ninguna señal de multicamino en la señal entrante. A continuación se compara el perfil de R_x con un perfil de correlación ideal. Si el error residual sobrepasa un umbral, significa que es muy probable que exista una señal de multicamino, y se empleará el modelo de una única señal de multicamino para estimar los parámetros. Una vez aplicado el modelo, si aún así el error residual mínimo supera ampliamente el umbral, será necesario desarrollar un modelo más complejo. Los principales inconvenientes de MMT son la elevada frecuencia de muestreo y consumo de potencia requeridos, y su sensibilidad al ruido, por lo que necesita un mayor tiempo de integración para proporcionar resultados válidos, y por tanto no puede funcionar en entornos multicamino dinámicos.

- *Vision Correlator*: supone una nueva aproximación para mitigar el multicamino completamente distinta al resto. Esta técnica se basa en el concepto de MMT, una variante de MEDLL. El principio fundamental del VC es medir con gran detalle las propiedades de radio frecuencia de las transiciones de chip. El VC se genera filtrando todas las transiciones de chip durante un periodo de tiempo. La Figura 2.7 ilustra el hardware necesario para implementar VC. Este avanzado hardware filtra el ruido por sobre-imposición de sucesivas transiciones de chip durante un intervalo específico de tiempo para dar lugar a un perfil de transición de chip medio. De esta forma se obtiene un vector de medidas discretas complejas, denominadas VS *(Vision Samples)*, que representan el perfil de transición. Estas muestras son procesadas posteriormente mediante MMT. MMT es un proceso de máxima verosimilitud optimizado que intenta estimar el mejor ajuste del vector VC *(Vision Correlation)* con múltiples versiones de una función de perfil de transición de referencia, siendo la primera de ellas la correspondiente a la señal directa, y el resto correspondientes a las posibles señales de multicamino. A pesar de que MMT puede emplearse para mitigar más de una señal de multicamino, en [9] se trata únicamente con una señal de multicamino. Una vez se ha establecido la función de perfil de transición de referencia, se emplea el algoritmo MMT para descomponer la señal VC *(Vision Correlation)* en señal directa y de multicamino. Posteriormente se calculan las amplitudes y ángulos de la fase de la portadora de las señales.

  La técnica VC proporciona una significativa mejora, en la detección y eliminación de señales de multicamino, sobre el resto de técnicas de mitigación de multicamino (*Narrow Correlator*, *Strobe & Edge Correlator*, *Gated Correlator*, etc). Es particularmente sensible a retardos del multicamino inferiores a 0.1 chip. Cuanto mayor es el ancho de banda y la frecuencia de muestreo del proceso Vision, más efectivo es éste.

- *Coupled Amplitude Delay Lock Loop*: este método surge como solución a los inconvenientes que

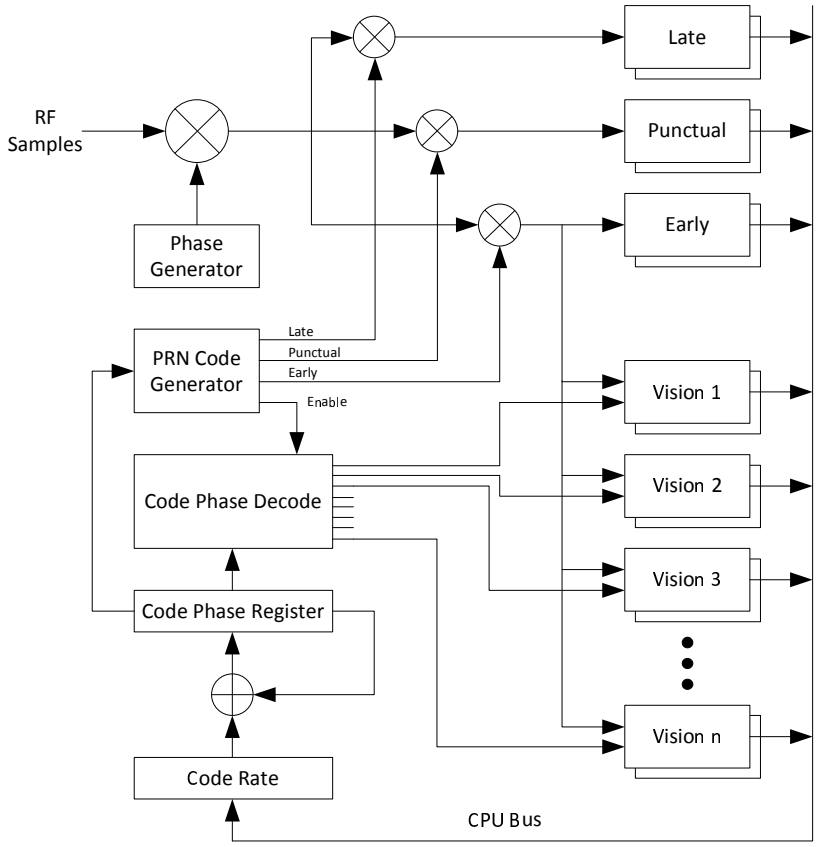


Figura 2.7: Diagrama de bloques de Vision Correlator.

presentan MEDLL y MMT (frecuencia de muestreo y consumo de potencia elevados). CADLL propone estimar los parámetros de las señales de multicamino de forma iterativa y realimentada, pero sin necesitar ningún bloque específico de estimación del multicamino. Además debe proporcionar una gran tolerancia al ruido y trabajar en condiciones normales de SNR con un tiempo de integración moderado. Por último, CADLL puede ajustar su estructura en función del número de señales de multicamino, y así mitigar mejor los multicaminos de pequeño retardo. Para alcanzar estos objetivos, CADLL parte de una arquitectura TurboDLL e innova MMT.

Los componentes fundamentales de la arquitectura CADLL son el ALL *(Amplitude Lock Loop)* y el DLL *(Delay Lock Loop)*, que estiman y siguen la amplitud de la señal y la fase del código respectivamente. La estructura del bloque ALL se basa en un bucle de realimentación muy similar a un DLL o PLL que puede estimar y seguir la evolución de la amplitud de la señal deseada. El bloque ALL está formado por un estimador inicial, y por un filtro (paso-bajo para reducir los efectos del ruido) junto con un integrador en el bucle de realimentación. Cada par ALL-DLL da lugar a una Unidad, de manera que la estructura CADLL dispone de M+1 Unidades para separar y seguir la señal directa además de las M componentes de multicamino. La Figura 2.8 representa el diagrama de bloques de la arquitectura CADLL. Como se puede observar, la señal de entrada a cada Unidad es la señal entrante $S_d(t)$ menos las señales de salida del resto de Unidades. La señal de salida de cada Unidad es en realidad una estimación de una de las componentes de la señal entrante. Las características de realimentación de la estructura CADLL permiten que las estimaciones converjan en los valores correctos tras un periodo de integración. El proceso realizado por CADLL es esencialmente un ajuste no lineal a la señal entrante por M+1 señales de código ensanchado. En la práctica CADLL puede ser visto como un estimador de señal no lineal.

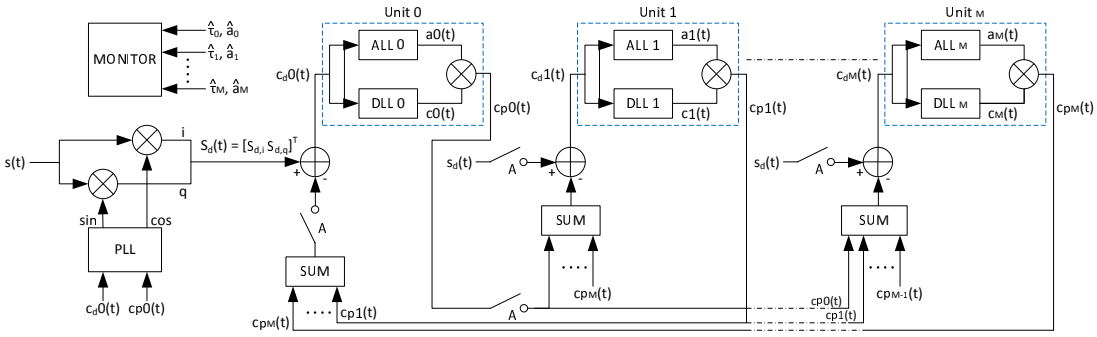


Figura 2.8: Diagrama de bloques de CADLL.

La arquitectura CADLL dispone de la configuración inicial de seguimiento estándar, y de la configuración de seguimiento CADLL. La primera de ellas, formada por un PLL y un DLL, es empleada para generar el estado inicial del que parte la segunda y que así converja en un estado final correcto. El DLL empleado en la etapa inicial es el DLL_0, cuyo estado inicial para la señal directa se encuentra muy próximo al verdadero retardo de la señal directa. La estimación de la amplitud de la señal directa es proporcionada por ALL_0 trabajando en la configuración inicial. De este modo se determina un estado inicial adecuado para la Unidad_0. Debido a que la señal directa tiene los efectos más significativos de todas las componentes, se garantiza la estabilidad y convergencia de todo el sistema.

La principal tarea del bloque de monitorización es monitorizar el seguimiento del estado de todas las Unidades para que el número de éstas coincida con el número de componentes de multicamino. Cuando una componente de multicamino no está presente en una Unidad, ésta sigue ruido o reflexiones irrelevantes, lo cual provoca que la amplitud estimada sea muy pequeña y que la fase del código varíe aleatoriamente. Para hacer frente a este problema el bloque de monitorización calcula periódicamente dos valores: la diferencia de retardos de código media entre cada par de Unidades consecutivas y la amplitud media de cada unidad. Si cualquiera de estos valores se encuentra por debajo de un umbral establecido la Unidad se apaga. Además, este bloque mantiene el orden de seguimiento de cada Unidad (de forma que la Unidad_0 siga la señal directa, la Unidad_1 siga el multicamino más cercano, ...) empleando la técnica denominada proyección en frecuencia. Con esta técnica se fuerza a que la diferencia de retardos de código entre dos Unidades consecutivas no baje de un cierto umbral de retardo. En caso de que esta diferencia decaiga por debajo de dicho umbral se genera una corrección en frecuencia del código.

En [10] se ha demostrado que el rendimiento de CADLL frente a ruido es superior respecto a cualquier otro método de estimación del multicamino. Además, en mitigación de multicaminos su rendimiento incluso supera al de HRC, pues tanto para multicaminos en fase como fuera de fase, el área de error de seguimiento en CADLL es mucho menor que en HRC. A diferencia de CADLL, para retardos de aproximadamente 1.0 chip HRC incrementa significativamente el error. Otros factores que afectan al rendimiento de CADLL son el ancho de banda del filtro pre-correlación y el espaciado de la correlación E-L. Cuando se emplea un correlador estrecho el ancho de banda del filtro pre-correlación no puede ser inferior al inverso del espaciado, es decir, el espaciado depende del ancho de banda. Hay que tener en cuenta que, cuanto menor es el espaciado menor es el área de error para multicaminos de retardo pequeño, pero mayor ancho de banda se requiere. Por último, la capacidad de adaptarse a diferentes entornos de multicamino es proporcional a la complejidad del sistema, por lo que es necesario alcanzar un compromiso en función de los recursos hardware y software del receptor.

El mayor inconveniente de la arquitectura CADLL es la incapacidad de estimar información de fase, lo cual impide la mitigación de multicaminos en la fase de la portadora. Ello se debe a que el ALL del que dispone cada Unidad únicamente estima la componente I *(In-phase)* de la amplitud, y por tanto sólo se cancela dicho canal. Esto hace que la arquitectura sea menos compleja, pero al mismo tiempo limitada, pues se pierde la información de la componente Q *(Quadrature)* de la amplitud. En ocasiones, como cuando la potencia de las componentes de multicamino es mayor que la potencia de la señal directa, la energía se concentra en el canal Q, y al no disponer de dicha información, la estructura CADLL no funciona correctamente.

- *Enhanced Coupled Amplitude Delay Lock Loop*: representa una mejora de la arquitectura CADLL con el objetivo de solventar los inconvenientes que ésta presenta [10]: a) problemas de convergencia en función de los valores iniciales cuando la arquitectura dispone de más de dos Unidades; b) CADLL no puede mitigar los errores de fase en la portadora ya que no puede estimar información de fase. Para hacer frente a estos problemas se ha redefinido la arquitectura CADLL tal como se muestra en la Figura 2.9, añadiendo a cada Unidad un bloque ALL extra. De esta forma, ahora cada Unidad puede estimar las componentes I/Q *(In-phase/Quadrature)* de la amplitud, y por consiguiente, realizar la operación de cancelación en ambos canales.

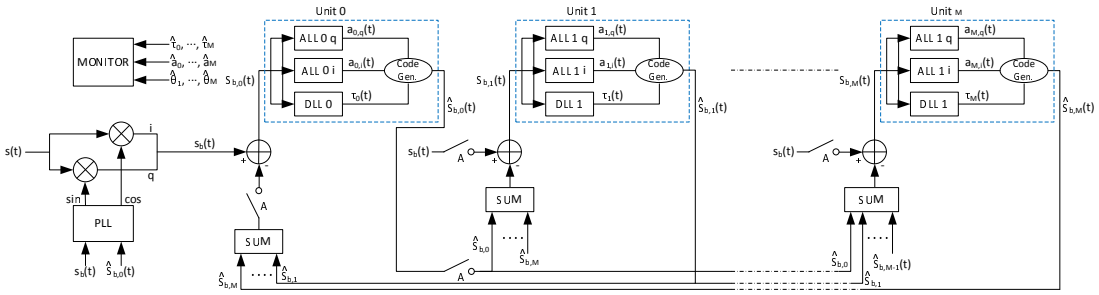


Figura 2.9: Diagrama de bloques de ECADLL.

El principio de funcionamiento de ECADLL es exactamente el mismo que el de CADLL. Presenta un excelente rendimiento en la mitigación de multicaminos tanto en la fase del código como de la portadora en comparación con otras técnicas como HRC o el correlador estrecho. En el caso de entornos en los que las señales de multicamino pueden tener una amplitud superior a la de la señal directa, ECADLL consigue separarlas, anulando así el error en la fase del código. A diferencia de ECADLL, otros métodos tratan la señal directa como si fuese de multicamino, cancelándola e incrementando el error en la fase del código a medida que el retardo del multicamino aumenta.

2.2.2 Mitigación del multicamino en UWB

La tecnología UWB proporciona una solución de bajo coste para aplicaciones de posicionamiento en espacios interiores en las que se requiere una elevada precisión. La técnica más común de estimación de la posición de un móvil basado en UWB es emplear el ángulo de llegada (AOA[4]) de las señales a dos o más estaciones base de referencia. El AOA realmente es estimado a partir de la diferencia de tiempos de llegada (TDOA[5]) de las señales a dos o más antenas en cada estación base. Para la estimación del TDOA entre antenas en una estación base, generalmente se realiza una correlación cruzada entre las señales en los canales recibidos, seguido de una detección de pico en la correlación cruzada. Sin embargo, este método presenta serios problemas cuando el SNR es bajo, o en presencia de multicaminos o interferencias

[4] AOA: Angle-of-Arrival.
[5] TDOA: Time-Difference-of-Arrival.

de banda estrecha. Cuando este tipo de interferencias aparecen dentro de la ventana de correlación la estimación de tiempo puede llegar a ser totalmente errónea. A continuación se describen dos métodos para tratar de resolver los problemas mencionados anteriormente.

- En [15] se presenta un algoritmo eficiente basado en un procesamiento de señal temporal-frecuencial, cuyo diagrama de bloques se muestra en la Figura 2.10. Las dos primeras etapas del algoritmo propuesto consisten en bancos de filtros digitales paso-banda, y una sección de detección SNR/SIR[6]. Tras estas dos etapas se identifican las sub-bandas de frecuencia de la señal con buen SNR/SIR. Estas sub-bandas son empleadas para estimar el TDOA entre las dos antenas receptoras. Empleando únicamente las sub-bandas con buen SNR/SIR se consigue eliminar gran parte del ruido y de las interferencias presentes en la señal original recibida.

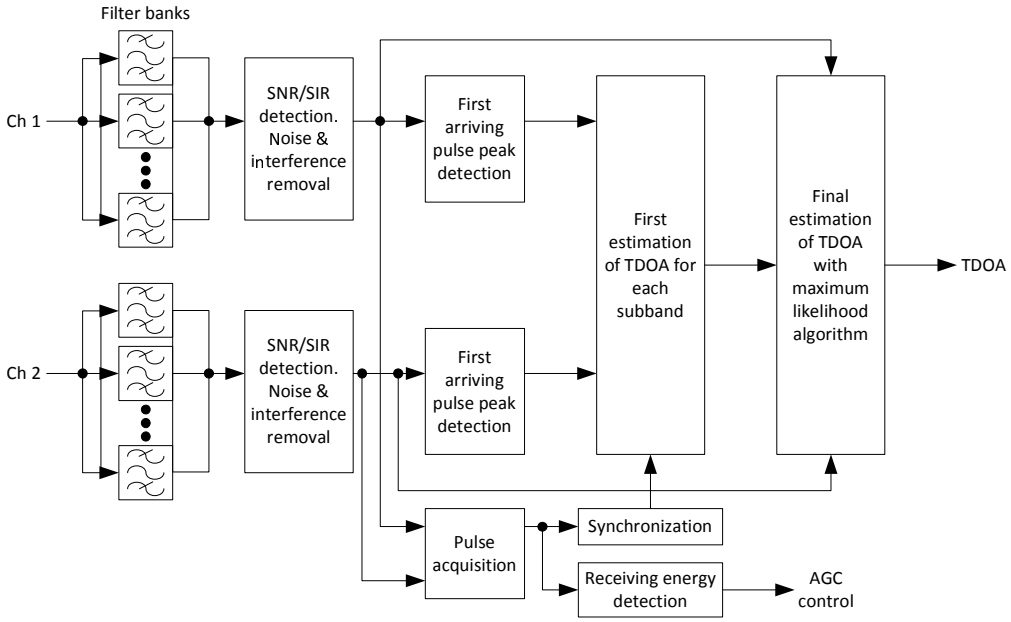


Figura 2.10: Diagrama de bloques del algoritmo de dominio temporal-frecuencial para la estimación de TDOA en UWB.

En el método convencional de realizar la correlación cruzada y detección de pico en el dominio temporal, el multicamino más cercano dentro de la ventana de correlación cambiará el pico de la correlación, provocando que la información temporal sea errónea. Para evitar este problema, en la tercera etapa del algoritmo se emplea la detección de la llegada del primer pulso y el seguimiento del pico. De esta forma no se necesita realizar la correlación cruzada, ya que cuando se detecta el primer pico se termina la búsqueda, y la información de fase en el pico es registrada para el cálculo temporal. Ello permite también no emplear la última parte de la señal, que es donde puede residir la señal de multicamino. Después de la detección de la llegada del primer pico se calcula la primera estimación de TDOA entre las dos antenas para todas las sub-bandas. Basada en la primera estimación de TDOA, la última estimación de TDOA se realiza aplicando un algoritmo de ML en las señales sub-banda procesadas.

El algoritmo propuesto basado en el dominio temporal-frecuencial presenta un rendimiento claramente superior respecto al algoritmo convencional de correlación cruzada en el dominio temporal en condiciones de bajo SNR, y en presencia de interferencias en banda o fuertes multicaminos. El procesamiento temporal-frecuencial es usado debido a que permite extraer la información útil de la señal excluyendo las sub-bandas con elevada contaminación de ruido e interferencias.

[6]SIR: Signal-to-Interference Ratio.

- En sistemas de medida a distancia basados en UWB, la distancia entre emisor y receptor normalmente se obtiene mediante la estimación del tiempo de llegada (TOA[7]) de la señal recibida. En situaciones en las que emisor y receptor no tienen línea de visión (NLOS[8]), la componente directa de la señal tiene que viajar a través de una o más obstrucciones que causan un exceso de retardo, por lo que la estimación del TOA experimenta un offset positivo, que es la principal fuente de error en medidas a distancia. Por este motivo, en [16] se propone un novedoso método de mitigación NLOS basado en comparar el tiempo de llegada y la energía recibida de la señal directa y de las señales de multicamino. Empleando el modelo clásico de pérdidas del camino de propagación de la señal, la expresión de la estimación del error NLOS es deducida y modificada teniendo en cuenta los efectos de la dirección de la antena.

  Los sistemas de medida de distancias UWB basados en la estimación del TOA normalmente obtienen el tiempo de propagación de la señal estimando el TOA de la componente directa. Las obstrucciones que la componente directa de la señal encuentra durante su propagación no sólo causan un exceso de retardo, sino también de atenuación. En entornos multicamino la señal sufre una atenuación adicional debido a los efectos de reflexión y difracción. El cálculo de la energía recibida procedente de la componente directa debe tener en cuenta el exceso de atenuación causado por las obstrucciones que se encuentre. Se ha considerado un modelo compuesto por dos caminos de propagación, el directo y el reflejado. Asumiendo que la antena de transmisión es idealmente omnidireccional, los caminos directo y reflejado presentan igual energía recibida en el punto de referencia. Se ha definido una métrica denominada *conf* basada en el producto de la amplitud al cuadrado por el TOA elevado al exponente de pérdidas del camino de propagación. Si se emplea el camino que presenta mayor valor de confianza (MCP[9]) la estimación del grosor total de las obstrucciones, y por tanto de la atenuación total, será más precisa.

  La deducción anterior asume que las antenas son perfectamente omnidireccionales, lo cual es difícil de garantizar en la práctica. Para calibrar los efectos de la dirección de la antena se ha definido un transmisor omnidireccional virtual, y se ha asumido que todos los caminos de llegada son radiados desde éste. El offset de localización que existe entre la antena real y virtual determina que es necesario calibrar los valores de TOA originales. Empleando los valores de TOA calibrados en vez de los originales se consigue una estimación de distancia más precisa.

  Cuando el pulso transmitido llega al receptor, la señal recibida está compuesta por la componente directa, las componentes de multicamino, ruido, e interferencias. Mediante la búsqueda del pico de correlación se obtienen el TOA y la amplitud del camino más fuerte. A continuación se normaliza la amplitud de la señal recibida y se desfasa en el tiempo respecto a los valores amplitud y TOA del pico de correlación. Para detectar la componente directa de la señal sólo se tiene en consideración la porción de señal que incluye los caminos que llegan antes que el camino más fuerte y el propio camino más fuerte. Puesto que el TOA del camino más fuerte ya ha sido obtenido, para estimar el TOA del camino directo simplemente hay que estimar la diferencia de TOA's entre el camino directo y el más fuerte; esta estimación se realiza mediante un algoritmo GML *(Generalized Maximum Likelihood)*. La detección del camino de máxima confianza de la señal se realiza siguiendo el mismo principio usado en la detección del camino directo, pero en este caso sólo se tiene en cuenta la porción de señal que incluye los caminos que llegan más tarde que el camino más fuerte.

[7] TOA: Time-of-Arrival.
[8] NLOS: Non-Line-of-Sight.
[9] MCP: Maximum Confidence Path.

2.2.3 Mitigación del multicamino en interferometría

La interferometría heterodina es una técnica ampliamente usada para medir desplazamientos de un orden inferior a nanometros. Las interferencias espúreas limitan el rendimiento de las medidas de interferometría. Estas interferencias pueden originarse a partir de diversas fuentes: luz dispersa, fugas espaciales o de polarización, y diafonía en el sistema de adquisición de datos. Todos estos mecanismos corresponden a interferencias por multicamino: señales procedentes de la misma fuente llegan a un punto de medida a través de diferentes caminos. A continuación se describen tres técnicas para mitigar este tipo de interferencias en sistemas de interferometría.

- En [17] se propone mejorar la interferometría digitalmente como una forma de aislar las señales interferométricas basada en sus retardos. Las señales interferométricas son etiquetadas en el tiempo mediante la modulación en fase de la fuente láser con un código PRN *(PseudoRandom Noise)*. La técnica propuesta mejora la sensibilidad en las medidas explotando las propiedades de autocorrelación del código PRN para aislar únicamente las señales de interés y rechazar las interferencias espúreas. Las propiedades del código PRN determinan el grado de aislamiento. A diferencia de las medidas basadas en código PRN estándar, la interferometría mejorada digitalmente mantiene la sensibilidad interferométrica completa determinada por la longitud de onda óptica. Otro beneficio del aislamiento de señal basado en su retardo es la habilidad de medir múltiples componentes ópticos con un único sistema de medida.

  En la Figura 2.11 se representa un interferómetro heterodino mejorado digitalmente empleado para medir cambios en la separación de tres espejos parcialmente reflectivos. El haz de láser emitido es separado en dos: el primer haz es dirigido hacia un modulador electro-óptico (EOM[10]), y el segundo es dirigido a un oscilador local. El oscilador local está desplazado en frecuencia por un modulador acústico-óptico (AOM[11]) que proporciona en el fotodetector una señal heterodina de frecuencia f_h. El haz láser que llega al EOM es modulado por un código PRN, que mantiene o invierte la fase del haz, antes de que sea dirigido hacia los tres espejos. La luz reflejada por estos espejos es recombinada con la luz procedente del oscilador local, de forma que la señal interferente es medida por el fotodetector. La modulación PRN proporciona a cada señal un desplazamiento de fase variante en el tiempo único para su tiempo de vuelo procedente del EOM. Además, invierte aleatoriamente la amplitud de la señal heterodina medida por el fotodetector.

  Para recuperar la señal heterodina la salida del fotodetector es multiplicada por el mismo código PRN retardado una cantidad igual al retardo opto-electrónico total de la reflexión de interés. Con los retardos coincidentes, esta operación revierte la inversión de las señales procedentes de los espejos recuperándose en forma de señales heterodinas estándar. Se dice que la información de cada señal es desensanchada en una sinusoide pura a la frecuencia heterodina f_h. La fase de cada señal heterodina contiene información sobre el desplazamiento de su respectivo espejo. Las reflexiones en las que los retardos de modulación y demodulación difieren en más de 1.0 chip de código no son desensanchadas, sino que son reinvertidas aleatoriamente y aparecen como ruido de banda ancha de fondo en la medida. Este ruido puede ser rechazado con un mediado apropiado en el fasómetro. Finalmente, la información sobre la separación de los espejos es obtenida diferenciando la fase de las señales decodificadas.

  Debido a que el aislamiento entre las señales interferentes depende de las propiedades del código PRN, la frecuencia de chip del código f_{chip} determina la diferencia de retardo mínima requerida para un aislamiento óptimo. Cuando f_h es un múltiplo entero de f_{chip} se logra una supresión mucho

[10] EOM: Electro-Optic Modulator.
[11] AOM: Acousto-Optic Modulator.

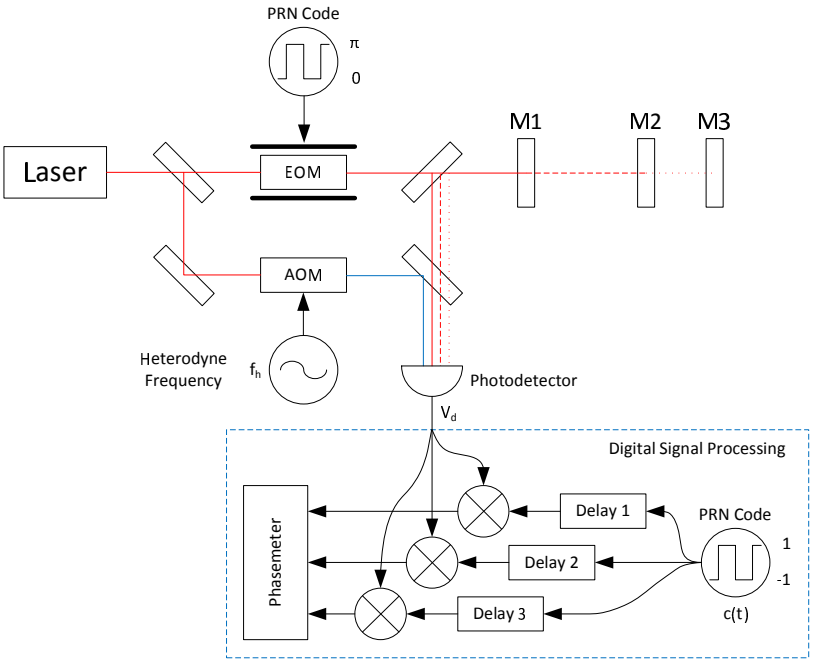


Figura 2.11: Interferómetro heterodino mejorado digitalmente.

mejor. He aquí un compromiso entre el aislamiento y la velocidad de medida, pues para obtener mayor supresión se requiere incrementar la longitud de los códigos.

Los errores en el retardo de decodificación no producen directamente error de fase sino errores de amplitud en las señales heterodinas decodificadas, incrementándose la contaminación por señales indeseadas. Para un rendimiento óptimo el error en el retardo de decodificación debe ser muy inferior a $1/f_{chip}$.

Un beneficio importante de multiplexar señales a través de un único sistema de medida es el rechazo al ruido en modo común procedente de varias fuentes: retardos en la fase causados por el fotodetector o cambios en la latencia del conversor analógico-digital (ADC[12]) (puesto que se usa un único fotodetector y ADC). Las interferencias electrónicas y las diafonías son también rechazadas debido a que las señales permanecen codificadas hasta que alcanzan el dominio digital.

- En [18] se propone una nueva arquitectura para medidas de desplazamiento láser en la que el procesamiento de señal adicional permite una drástica reducción en el tamaño y complejidad de la cabecera óptica. Esta técnica, denominada metrología acotada en distancia, aísla los retornos objetivo y de referencia mediante la codificación PRN del haz de medida. Las señales son separadas basándose en la diferencia de su retardo de propagación, dando lugar a una cabecera óptica altamente simplificada y compacta, y fácilmente integrable en un sistema de medida.

  En un sistema de interferometría heterodina estándar el haz láser emitido es dividido en las ramas de medida y local, estando desplazadas en frecuencia la una de la otra por f_{het}. En la arquitectura propuesta, mostrada en la Figura 2.12, además del desplazamiento de frecuencia f_{het} introducido por el AOM, un EOM modula con un código PRN el haz que viaja por la rama de medida manteniendo o invirtiendo su fase. La luz modulada es dirigida mediante un circulador a la cabecera óptica y al retro-reflector objetivo. Los retornos objetivo y de referencia vuelven al circulador y son dirigidos hacia un único detector donde se mezclan con el haz procedente del oscilador local. La salida del detector es una superposición lineal de las señales objetivo y de referencia "downconverted". Cada

[12] ADC: Analog-Digital Converter.

una de ellas es una sinusoide de frecuencia f_{het} multiplicada por el código PRN de frecuencia f_{chip}. El código PRN tiene un retardo igual al tiempo de propagación que hay desde el modulador de fase hasta el detector. La señal deseada es extraída demodulando la salida del detector con el código PRN y su retardo apropiado. Cualquier componente de la salida del detector con un retardo diferente permanecerá como una sinusoide modulada aleatoriamente contribuyendo como ruido blanco en el proceso de medida de fase. En el experimento descrito, tanto la salida del detector como el código PRN son digitalizados por un osciloscopio digital y posteriormente transferidas a un ordenador donde se realiza la demodulación con el retardo de interés. Las amplitudes y fases de las sinusoides son extraídas determinando las componentes I/Q a la frecuencia f_{het} de cada salida demodulada. En la práctica, el procesamiento de señal puede realizarse en tiempo real empleando una FPGA *(Field Programmable Gate Array)*.

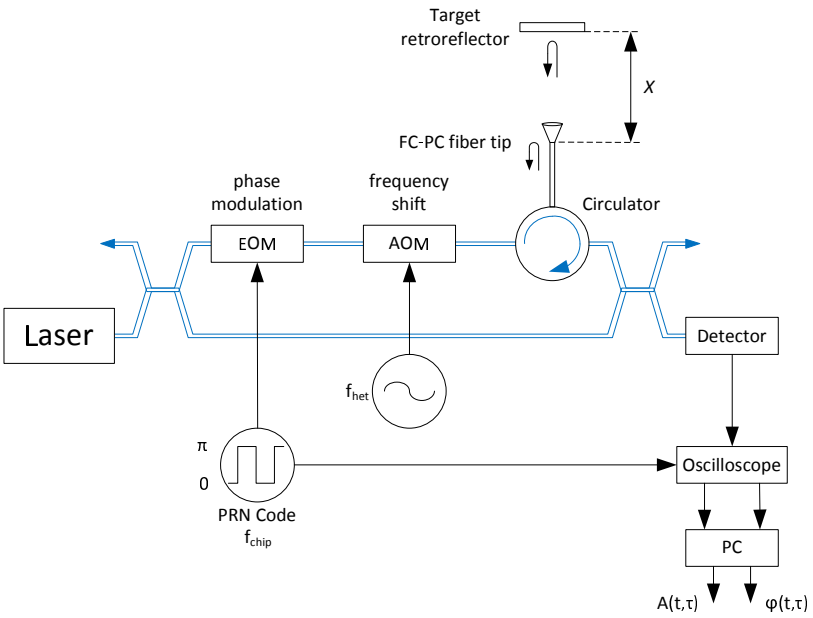


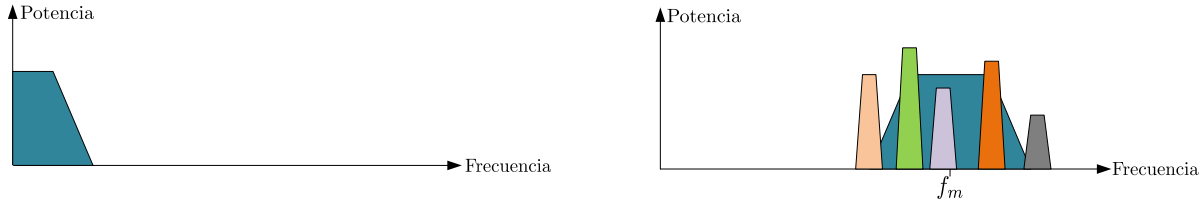
Figura 2.12: Metrología acotada en distancia.

Hay que destacar que la medida óptica de fase es robusta a las imperfecciones en la modulación y demodulación de las señales. Si la profundidad de la modulación de fase se desvía de π, el resultado es un ligero decremento en la amplitud de la señal demodulada, pero el valor de fase esperado permanece inalterado. Igualmente, el retardo del código PRN para la demodulación no tiene porqué coincidir exactamente con el retardo de propagación deseado. Siempre y cuando el retardo correcto esté dentro de un margen de error inferior a 1.0 chip de código, la medida óptica de fase es invariante a las fluctuaciones del retardo.

2.3 Sistema de medida diferencial de distancia con mitigación del multicamino

2.3.1 Fundamentos teóricos

Cuando se desea transmitir información que está en banda base (Figura 2.13(a)), normalmente ésta es modulada por una portadora de elevada frecuencia (f_m), desplazándose su espectro a dicha frecuencia (Figura 2.13(b)). El principal problema de la modulación es que la banda de altas frecuencias a la que se ha desplazado el espectro está llena de otras señales empleadas para comunicaciones que contaminan la información a transmitir, son las denominadas interferencias de banda estrecha.



(a) Información en banda base.

(b) Información modulada.

Figura 2.13: Modulación en frecuencia.

El espectro ensanchado *(spread spectrum)* es una técnica de modulación ampliamente usada en telecomunicaciones para la transmisión de datos por radiofrecuencia. Esta técnica consiste en ensanchar el espectro de la señal a transmitir de forma que su energía se disperse a lo largo de una banda de frecuencias muy amplia y superior a la que ocupaba originalmente la información. Para producir el ensanchamiento de la banda de frecuecias se emplean señales pseudo-aleatorias, señales que se caracterizan por tener un gran ancho de banda y propiedades estadísticas similares al ruido, y por lo que a menudo son denominadas códigos PRN *(PseudoRandom Noise)*. Por ello, la señal transmitida tiene características pseudo-aleatorias, y su demodulación en el receptor sólo es posible si se dispone de la misma señal pseudo-aleatoria empleada para su modulación. La señal ensanchada posee un conjunto de características de especial interés:

- La transmisión de señales con espectro ensanchado tiene una resistencia superior a las interferencias de banda estrecha que cualquier otro tipo de transmisión. Esta transmisión también es resistente a las interferencias por multicamino ya que son tratadas de forma similar a las interferencias de banda estrecha (sólo si el retardo entre componentes cumple ciertas condiciones).
- Debido a la dispersión espectral de energía, el nivel de densidad potencia de la señal se reduce significativamente, haciendo que su detección sea muy complicada.
- Las características pseudo-aleatorias de la señal permiten que, aunque ésta haya sido detectada, sea incompresible si no se conoce cuál fue señal ensanchadora.
- Se puede transmitir varias señales de espectro ensanchado simultáneamente por el mismo medio si se emplean distintas señales pseudo-aleatorias incorreladas entre sí para producir el ensanchamiento. Esto da lugar a una técnica de acceso múltiple al medio denominada CDMA *(Code Division Multiple Access)*.

Los códigos PRN son secuencias de bits aparentemente aleatorias pero determinísticas que por sí mismos no aportan ninguna información. Estas secuencias son repetidas periódicamente y poseen una

función de auto-correlación muy similar al ruido blanco. Cuantos más bits se empleen en la generación de un código PRN más duración, ancho de banda, y aleatoriedad se conseguirá. Es muy común referirse a los elementos del código PRN como chips. El ensanchamiento producido sobre el espectro de la señal portadora es del orden del número de chips N_{chip} del código PRN, el cual depende del número de bits con que se genera el código: $N_{chip} = 2^{bits_{PRN}} - 1$.

Existen dos técnicas principales de espectro ensanchado: espectro ensanchado por secuencia directa, y espectro ensanchado por saltos de frecuencia. La primera de ellas, denominada DSSS *(Direct Sequence Spread Spectrum)*, consiste en modular la señal portadora de banda estrecha mediante un código PRN, y posteriormente en el receptor demodularla con una réplica generada localmente del código PRN. En cambio, la segunda técnica, denominada FHSS *(Frequency Hopping Spread Spectrum)* se basa en hacer saltar la frecuencia de la señal portadora de acuerdo con un código PRN; en el receptor se rastrean los cambios en frecuencia producidos para dar lugar a una señal de frecuencia intermedia constante. La técnica DSSS es la que emplea el GPS y en la que se basa este trabajo.

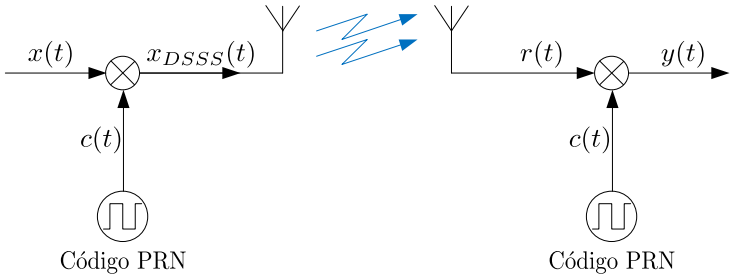


Figura 2.14: Diagrama de bloques de sistema de modulación/demodulación DSSS.

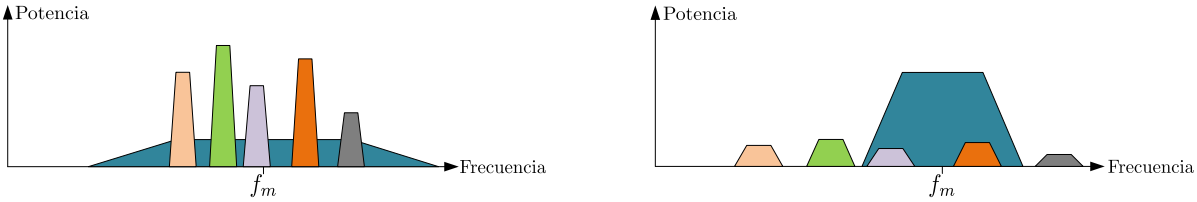
En la Figura 2.14 se representa un diagrama de bloques de un sistema genérico de modulación/-demodulación DSSS. La señal modulada en frecuencia a transmitir es $x(t)$, la cual es ensanchada por un código PRN $c(t)$ (2.3), obteniéndose $x_{DSSS}(t)$ (2.4). La señal recibida $r(t)$ (2.5), cuyo espectro se muestra en la Figura 2.15(a), está compuesta por la señal emitida $x_{DSSS}(t)$, AWGN *(Additive White Gaussian Noise)* $n(t)$, e interferencias de banda estrecha $x_i(t)$. Cuando $r(t)$ es multiplicada por el mismo código PRN $c(t)$ se obtiene la señal $y(t)$ (2.6), que presenta el espectro mostrado en la Figura 2.15(b). La demodulación de la señal $r(t)$ produce el desensanchamiento de $x_{DSSS}(t)$ y el ensanchamiento de $x_i(t)$. De esta forma se recupera la señal original $x(t)$ mientras que la potencia de las interferencias de banda estrecha es dispersada. El AWGN, al ser una señal aleatoria de banda ancha, no se ve afectado por el ensanchamiento.

$$c(t) = \sum_{n=1}^{N_{chip}} c[n] \cdot \prod \left(\frac{t - \frac{T_{chip}}{2} - n \cdot T_{chip}}{T_{chip}} \right), \qquad c[n] = \pm 1 \tag{2.3}$$

$$x_{DSSS}(t) = x(t) \cdot c(t) \tag{2.4}$$

$$r(t) = x_{DSSS}(t) + n(t) + x_i(t) \tag{2.5}$$

$$\begin{aligned} y(t) &= r(t) \cdot c(t) \\ &= x_{DSSS}(t) \cdot c(t) + n(t) \cdot c(t) + x_i(t) \cdot c(t) \\ &= x(t) + n(t) + x_i(t) \cdot c(t) \end{aligned} \tag{2.6}$$



(a) Información modulada ensanchada.

(b) Información modulada desensanchada.

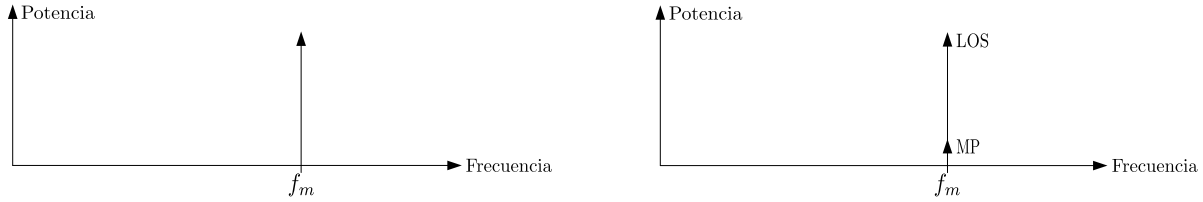
Figura 2.15: Modulación en frecuencia con espectro ensanchado.

Los satélites que conforman el sistema GPS emiten señales portadoras microondas de dos frecuencias distintas: L1 (1575.42 MHz) y L2 (1227.60 MHz). Para desplazar la fase de las portadoras L1 y L2 se emplean tres tipos de códigos binarios: C/A *(Coarse/Acquisition)*, P *(Precission)*, y NM *(Navigation Message)*. El código C/A es un código PRN repetitivo de 1.023 Mchips/s que modula la portadora L1 ensanchando su espectro por encima de 1 MHz de ancho de banda. Cada uno de los satélites modula la portadora con un código C/A diferente, código que al mismo tiempo actúa de identificador de cada satélite. Esta técnica de multiplexación por división de código permite la identificación de los satélites aunque todos ellos emitan la misma banda de frecuencia (L1), ya que el generador de códigos C/A produce una secuencia de 1023 chips diferente para cada uno de ellos. La medida de la distancia entre un satélite y un receptor se basa en la medida del tiempo de vuelo (t_d) de la señal L1 emitida por dicho satélite, y la velocidad de propagación c de ésta: $d = c \cdot t_d$. La medida del t_d se realiza sincronizando el código C/A del receptor con el código C/A de la señal de la señal recibida. El tiempo inicial que tarda en sincronizarse el código C/A en el receptor proporciona el TOA del código C/A entre el satélite y el receptor. Dado que el tiempo t_1 que tarda en sincronizarse el código C/A en el satélite es conocido, t_d es calculado a partir de t_1 y el TOA.

La modulación con espectro ensanchado no sólo se emplea en aplicaciones de transmisión de datos por radiofrecuencia, sino también en aplicaciones de medida de distancia dadas sus buenas características en la mitigación de interferencias por multicamino.

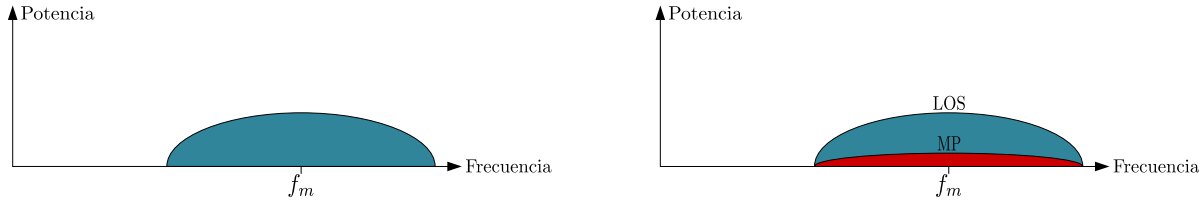
Como ejemplo de las propiedades de mitigación del multicamino de la técnica DSSS supóngase un sistema de medida de distancias que obtiene dichas medidas a partir del TOA de la señal emitida por el dispositivo emisor (US, IR, ...) hasta que alcanza el dispositivo receptor. Si se asume que la señal emitida es un tono como el representado en la Figura 2.16(a) de potencia 1.0 W, debido a las reflexiones que ésta sufre en el entorno, la señal recibida está formada por la componente que llega por el camino directo LOS *(Line-of-Sight)* entre emisor y receptor, y por una o varias componentes de multicamino MP *(MultiPath)*; por cuestiones de sencillez y simplificación se asume una única componente MP, por lo que el espectro de la señal recibida es el mostrado en la Figura 2.16(b). Asumiendo que la componente directa alcanza el receptor con una amplitud $\alpha_{LOS} = 1.0$ W y con un retardo $\tau_{LOS} = 1.0$ s, mientras que la componente de multicamino, a causa de la reflexión, lo hace con una amplitud $\alpha_{MP} = 0.1$ W y con un retardo $\tau_{MP} = 2.0$ s, la componente MP provoca que el TOA estimado por el sistema sea de $\hat{\tau_L} = 1.2$ s, cuando idealmente debiera ser de 1.0 s. En este caso, la relación de potencia entre las componentes MP y LOS es $\alpha_{MP}/\alpha_{LOS} = 10\,\%$.

Para reducir la influencia de la componente MP se emplea la técnica DSSS descrita anteriormente. La señal emitida es modulada por un código PRN produciendo un ensanchamiento de su espectro como se observa en la Figura 2.17(a). Ahora, la señal captada en el receptor presenta el espectro mostrado en la Figura 2.17(b), en el que se puede ver como el espectro de la componente MP también está ensanchado. El desensanchamiento de la señal recibida se realiza mediante una réplica localmente generada del mismo código PRN pero desfasada el retardo estimado por el sistema $\hat{\tau}_L = 1.2$ s. El porcentaje de potencia



(a) Señal emitida. (b) Señal recibida.

Figura 2.16: Emisión-recepción sin espectro ensanchado.



(a) Señal emitida ensanchada. (b) Señal recibida ensanchada.

Figura 2.17: Emisión-recepción con espectro ensanchado.

recuperada de las componentes LOS y MP tras la demodulación DSSS se muestra en la Figura 2.18 suponiendo un $T_{chip} = 1.0$ s. Cuando el retardo estimado de la señal recibida ($\hat{\tau}_L$) es igual al de alguna de las componentes ($\tau_{LOS,MP}$) la componente en cuestión es desensanchada al 100 %; en cambio, cuando la diferencia entre el retardo estimado y el retardo de las componentes LOS o MP es superior a T_{chip} la componente se desensancha al 0 %; en situaciones intermedias se asume una aproximación lineal para el porcentaje de desensanchamiento. Como se puede ver, la demodulación de la componente LOS no es completamente coherente, y la demodulación de la componente MP no es completamente incoherente. El espectro de la señal recibida una vez demodulada tiene un aspecto semejante al de la señal recibida sin espectro ensanchado (Figura 2.16(b)), sin embargo, su amplitud sí que se ha modificado significativamente debido a los porcentajes de potencia recuperados tras el desensanchamiento. Ahora, la componente LOS tiene una amplitud $\alpha_{LOS} = 1.0 \cdot 0.8 = 0.8$ W, mientras que la componente MP tiene una amplitud $\alpha_{MP} = 0.1 \cdot 0.2 = 0.02$ W, por lo que la relación de potencia entre ambas es de $\alpha_{MP}/\alpha_{LOS} = 2.5\,\%$. Cuanto más precisa sea la estimación de τ menor será la diferencia $|\tau_{LOS} - \hat{\tau}|$ y mayor $|\tau_{MP} - \hat{\tau}|$, y por tanto, mayor potencia de la componente LOS y menor potencia de la componente MP se recuperará, minimizando así el efecto de la componente MP en cualquier medida realizada sobre la señal demodulada.

Figura 2.18: Porcentaje de potencia recuperada de las componentes LOS y MP tras la demodulación DSSS en función del error de retardo del código local.

2.3.2 Descripción de la arquitectura de medida

El LPS IR digital del que parte este trabajo ha demostrado un buen rendimiento logrando errores de posicionamiento inferiores a 2.5 cm. Sin embargo, las interferencias por multicamino debidas a los requerimientos de cobertura del emisor y receptores constituyen la fuente de error más crítica del sistema, afectando al error de posicionamiento en el orden de cm o dm en función de la geometría del entorno y sus propiedades reflectivas. Ello ha motivado el estudio de un nuevo esquema de medida de distancias basado en IR y adaptado para aplicaciones en espacios interiores. Este nuevo esquema de medida mejora la precisión de las medidas ópticas de distancia al reducir los efectos de las interferencias por multicamino.

La arquitectura de medida que a continuación se describe se basa en la combinación de algunas de las técnicas descritas anteriormente a lo largo de la sección 2.2. Se mezclan técnicas de interferometría (en las cuales el retardo es conocido) estimando el retardo mediante técnicas de DSSS aplicadas a GNSS's y basadas en DLL's. La definición de los DLL's se ha llevado a cabo teniendo presente las ventajas que ofrece el estrechamiento del espaciado de los correladores que lo conforman [2], estableciéndose un espaciado inferior a 1.0 chip.

En la Figura 2.19 se muestra el diagrama de bloques del sistema de medida diferencial de distancia basado en el IR. La señal emitida está modulada inicialmente en amplitud (AM) por un tono sinusoidal de frecuencia f_m, y a continuación, su espectro es ensanchado por una secuencia PRN mediante DSSS. La señal de espectro ensanchado que alcanza cada receptor es seguida dentro de una etapa de sincronización que explota las propiedades de correlación de la señal recibida; el retardo relativo obtenido es empleado para generar una réplica local en fase con la secuencia PRN recibida. La réplica localmente generada es usada para demodular la señal recibida desensanchando su espectro y devolviéndola a su frecuencia original f_m. Mediante una etapa de demodulación I/Q asíncrona se estima la fase de la señal demodulada. La estimación diferencial de fase es obtenida tras restar a la fase estimada en un receptor la estimación procedente de otro receptor. Finalmente, se proporciona la estimación diferencial de distancia aplicando las ecuaciones (2.1) y (2.2).

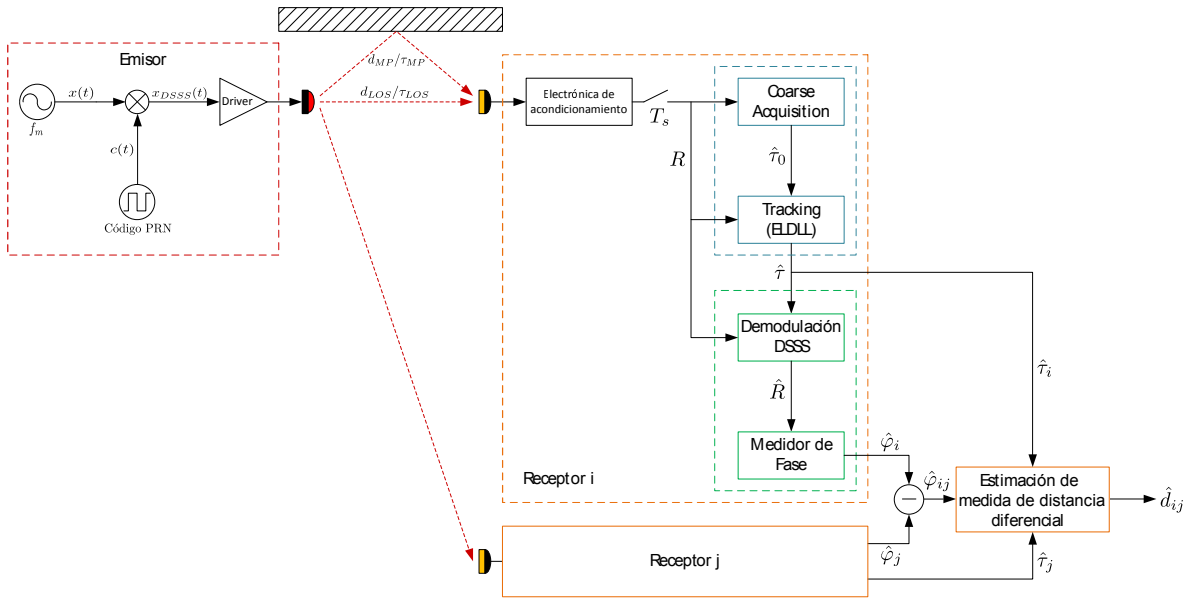


Figura 2.19: Diagrama de bloques de sistema de medida diferencial de distancia.

La capacidad de mitigación del multicamino de esta novedosa arquitectura recae en el proceso de demodulación DSSS. Como se vio al final del apartado 2.3.1, cuanto más próximo esté el retardo estimado para generar la réplica local de la secuencia PRN a la componente LOS, más coherente será la demodu-

lación que se le aplique a la componente LOS y menos coherente a las componentes MP. De esta forma se recuperará una mayor potencia de la componente LOS y una menor potencia de las componentes MP, por lo que se consigue reducir el error de multicamino en la estimación del retardo sobre la fase de la señal demodulada.

El error de multicamino depende tanto de la potencia como del retardo relativos de las componentes LOS y MP de la señal recibida. Normalmente, debido a la alta absorción de las superficies reflectantes del entorno, en situaciones de multicamino óptico en espacios interiores la componente LOS es la predominante en términos de potencia. Es por ello que el retardo estimado afectado por el multicamino está más próximo al valor real del retardo LOS que al retardo de la componente MP más rápida, excepto en situaciones de multicamino severas.

En [19] se presenta un modelo que relaciona la cantidad de potencia desensanchada en la banda AM en función del error de alineamiento entre la secuencia PRN de la señal recibida y la réplica localmente generada. Los resultados muestran que la potencia desensanchada está fuertemente relacionada con la función de correlación de la secuencia ensanchada, teniendo una relación lineal en el caso de que el error de alineamiento esté comprendido entre ± 1.0 chip, mientras que para una secuencia PRN concreta y errores de alineamiento superiores a T_{chip} presenta un valor pequeño.

La señal emitida $x_{DSSS}(t)$ (2.7) está formada por la componente AM sinusoidal $x(t)$ (2.8) modulada por la secuencia ensanchadora $c(t)$ (2.9) mediante DSSS. La componente AM $x(t)$ está definida por su frecuencia f_m y su fase φ_0, mientras que la secuencia PRN $c(t)$ es una repetición periódica de periodo $T_f = N_{chip} \cdot T_{chip}$ del código PRN $p(t)$ (2.10), donde T_{chip} es la duración de un chip del código PRN, y N_{chip} es el número de chips total del código PRN. Cada chip del código PRN está definido por una función ventana de duración T_{chip} (2.11).

$$x_{DSSS}(t) = x(t) \cdot c(t) \tag{2.7}$$

$$x(t) = \sin(2\pi f_m t + \varphi_0) \tag{2.8}$$

$$c(t) = \sum_{k=0}^{\infty} p(t - kT_f), \tag{2.9}$$

$$p(t) = \sum_{n=0}^{N_{chip}-1} p[n] \cdot \prod \left(\frac{t - nT_{chip}}{T_{chip}} \right) \qquad p[n] = \pm 1 \tag{2.10}$$

$$\Pi \left(\frac{t}{T_{chip}} \right) = \begin{cases} 1 & 0 \leq t \leq T_{chip} \\ 0 & \end{cases} \tag{2.11}$$

Los parámetros de diseño en cuanto a estructura de señal son la frecuencia de modulación AM f_m, la frecuencia de chip f_{chip}, y la relación de fase entre ambas determinada por φ_0. Los valores de estos parámetros han sido definidos en función del ancho de banda del enlace IR con el objetivo de maximizar la precisión tanto en la etapa de seguimiento como en la estimación final de distancia. Para maximizar el rendimiento en ambas etapas se han tenido en cuenta dos consideraciones: en primer lugar, maximizar el ancho de banda disponible, en términos de potencia, está directamente relacionado con el nivel de SNR de las señales recibidas, y por consiguiente con la precisión de ambas estimaciones; en segundo lugar, la frecuencia de chip está directamente relacionada con la precisión en la etapa de seguimiento, mientras que la frecuencia AM está directamente relacionada con la precisión en la estimación de fase, por lo

tanto, ambas deben ser lo más elevadas posible. En base a estas dos consideraciones, se ha establecido la frecuencia AM y de chip como la mitad del ancho de banda del canal IR. Por último, la relación de fase entre la componente AM y la secuencia ensanchadora debe ser elegida de forma que minimice la distorsión limitando la banda del canal en las transiciones de fase.

2.3.2.1 Sincronización C/A

La etapa de sincronización C/A tiene como función proporcionar una estimación gruesa del retardo de la señal entrante que actúe como valor inicial de enganche para el bucle de tracking ELDLL, o bien se reenganche en caso de haber perdido el enganche. Esta etapa se divide en dos procesos secuenciales. El primero de ellos es una búsqueda de los candidatos de retardo basada en los diferentes valores de correlación entre la señal entrante con la réplica localmente generada barriendo todas las posibles fases de ésta con una cierta resolución. Una vez finalizado este proceso se analizan los resultados de correlación obtenidos para detectar la presencia de un señal de entrada válida. Si la detección tiene éxito, la fase de la réplica localmente generada que proporciona el mayor valor de correlación es la estimación gruesa de retardo $\hat{\tau}_0$.

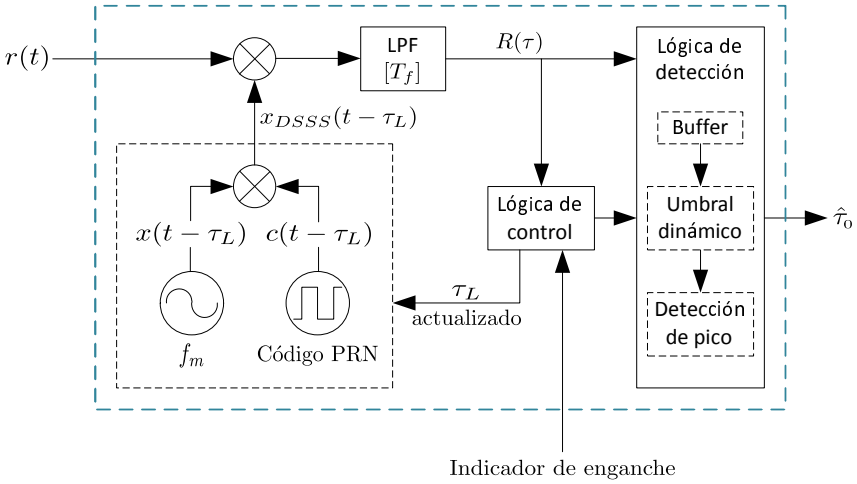


Figura 2.20: Diagrama de bloques de la etapa de sincronización C/A.

En la Figura 2.20 se representa el diagrama de bloques de la etapa de sincronización C/A. La señal entrante $r(t)$ es correlada con la réplica local de la señal esperada $x_{DSSS}(t)$ generada con una cierta fase τ_L cada periodo de correlación T_f (duración del código PRN $p(t)$). El resultado de correlación para una fase particular $R(-\tau_L)$ (2.12) es almacenado, y a continuación la lógica de control actualiza la fase τ_L con una resolución $\Delta\tau_L$. Cuando todos los valores de correlación han sido calculados y almacenados, la lógica de detección comienza a funcionar. Todos los valores de correlación son analizados para poder definir un umbral basado en la media de estos valores. Este umbral es usado por el detector de pico para determinar la presencia de una señal válida. En caso de una detección exitosa, la fase local τ_L asociada al máximo valor del pico detectado es llevada a la etapa de tracking ELDLL como el mejor retardo estimado $\hat{\tau}_0$.

$$R(\tau) = R(-\tau_L) = \frac{1}{T_f}\int_0^{T_f} r(t)\cdot x_{DSSS}(t-\tau_L)d\tau_L \tag{2.12}$$

El principal compromiso de diseño de esta etapa está relacionado con el error total en la estimación del retardo y el tiempo de adquisición medio. El hecho de emplear una mayor resolución en la fase local ($\Delta\tau_L$) implica una reducción del error en la estimación del retardo, debido a que la discretización de la correlación medida es menos severa, pero conlleva un incremento del tiempo de adquisición medio. Del

mismo modo, emplear longitudes de correlación mayores reduce el error añadido por el ruido AWGN, pero aumenta el tiempo de adquisición medio. En ambos casos, un mayor tiempo de adquisición implica un peor comportamiento dinámico.

2.3.2.2 Tracking ELDLL

La etapa de seguimiento del retardo tiene como función proporcionar una estimación continua del retardo de la señal entrante. El retardo estimado es usado en la siguiente etapa, explicada en el apartado 2.3.2.3, para generar una réplica local de la secuencia PRN esperada con la que realizar la demodulación DSSS parcialmente coherente de la señal de entrada. El retardo seguido es usado además en la etapa de medida diferencial de distancia final para solucionar la incertidumbre de ciclo de la estimación de medida de distancia basada en fase.

La precisión alcanzable en esta etapa depende de los mismos factores que condicionaban la precisión en la etapa de sincronización C/A. Sin embargo, hay una diferencia muy significativa, y es que en esta etapa no se realiza una búsqueda de candidatos de retardo, por lo que la resolución de la fase de la réplica local no influye en la dinámica del sistema. Ello permite que el compromiso entre los errores relacionados con la precisión y los errores dinámicos acumulados sea menos estricto, consiguiéndose así una mayor precisión global en la estimación.

El bucle de seguimiento se basa en un ELDLL *(Early-Late Delay Lock Loop)* de banda base. Este tipo de bucles de seguimiento, una vez enganchados, correlan la señal entrante con versiones ligeramente adelantadas *(Early)* y atrasadas *(Late)* de la réplica local para obtener información sobre el error de alineamiento de ésta con la señal de entrada. Dicha información es usada para corregir la fase local de forma que se mantenga el seguimiento y se proporcione una estimación continua del retardo.

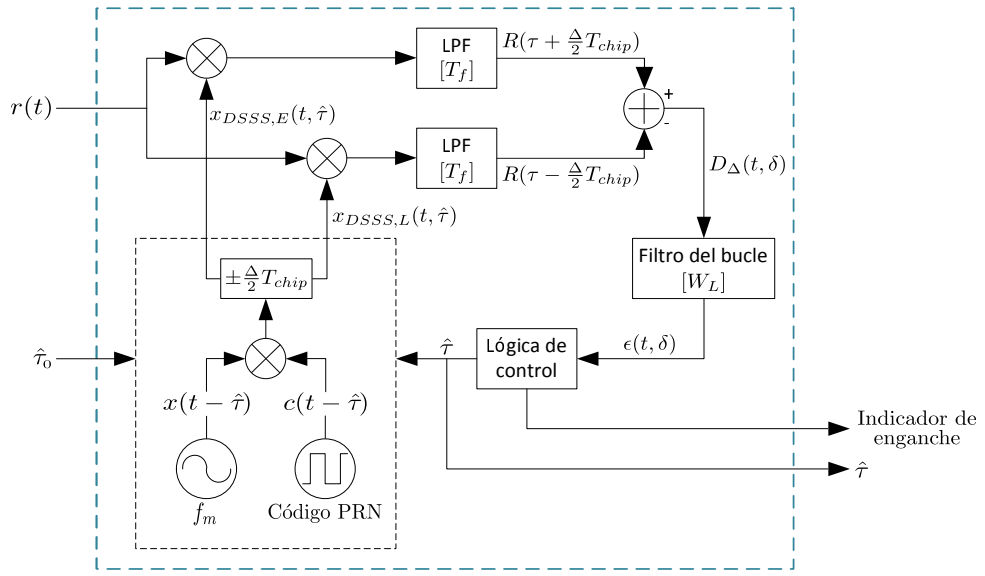


Figura 2.21: Diagrama de bloques de la etapa de tracking ELDLL.

La estructura de la etapa de seguimiento ELDLL se muestra en la Figura 2.21. Cuando la estimación gruesa del retardo $\hat{\tau}_0$ está disponible, ésta es usada para inicializar el generador de réplicas local. Las versiones *early* (2.13) y *late* (2.14) de la señal recibida esperada son generadas con una fase inicial $\hat{\tau}_0$ que, tras cerrar el lazo, será reemplazada por la estimación actual del retardo seguido $\hat{\tau}$. El retardo entre las señales *early* y *late* en relación a la estimación actual del retardo es determinado por el espaciamiento *early-late* establecido en Δ chips.

$$x_{DSSS,E}(t,\hat{\tau}) = x_{DSSS}\left(t - \hat{\tau} + \frac{\Delta}{2}T_{chip}\right) \tag{2.13}$$

$$x_{DSSS,L}(t,\hat{\tau}) = x_{DSSS}\left(t - \hat{\tau} - \frac{\Delta}{2}T_{chip}\right) \tag{2.14}$$

Las señales *early* y *late* son correladas con la señal de entrada $r(t)$ con un periodo de correlación T_f. Los valores de correlación resultantes R_E (2.16) y R_L (2.17) corresponden a los valores de la correlación de $r(t)$ con la réplica local actual $x_{DSSS}(t-\hat{\tau})$ (2.15) adelantados y retrasados $\Delta/2$. Debido a las propiedades de simetría de la función $R(\tau)$ los valores R_E y R_L tienden a ser iguales cuando el error de estimación tiende a 0. En función del signo del retardo, los valores R_E y R_L tienden a crecer con signo diferente; esta propiedad es explotada por el ELDLL para mantener el seguimiento del error de la estimación y corregirlo.

$$R(\tau) = R(-\hat{\tau}) = \frac{1}{T_f}\int_0^{T_f} r(t)\cdot x_{DSSS}(t-\hat{\tau})d\hat{\tau} \tag{2.15}$$

$$R_E = R\left(\tau + \frac{\Delta}{2}T_{chip}\right) \tag{2.16}$$

$$R_L = R\left(\tau - \frac{\Delta}{2}T_{chip}\right) \tag{2.17}$$

Los valores de correlación R_E y R_L son restados para dar lugar a la función del discriminador $D_\Delta(t,\delta)$ (2.18), siendo δ (2.19) el error de seguimiento normalizado. El discriminador tomará valores positivos ($R_E > R_L$) cuando $(\tau_{LOS} - \hat{\tau}) < 0$, por ejemplo, el retardo estimado es mayor que el actual, por lo que se dice que la réplica local actual está retrasada, estando el retardo de la réplica *early* más próximo al de la señal de entrada. En cambio, el discriminador tomará valores negativos ($R_E < R_L$) cuando $(\tau_{LOS}-\hat{\tau}) > 0$, por lo que la réplica local actual está adelantada, y el retardo de la réplica *late* estará más próximo al retardo de la señal de entrada. El perfil de la función del discriminador depende fuertemente del espaciado *early-late* Δ.

$$D_\Delta(t,\tau) = R\left(\tau + \frac{\Delta}{2}T_{chip}\right) - R\left(\tau - \frac{\Delta}{2}T_{chip}\right) \tag{2.18}$$

$$\delta = \frac{\tau_{LOS} - \hat{\tau}}{T_{chip}} \tag{2.19}$$

La salida generada por el discriminador es llevada al filtro del bucle, el cual define el ancho de banda (W_L) de la estimación final, y por consiguiente la contribución del ruido al error, limitando el ancho de banda de post-correlación tanto como la dinámica del sistema lo permita.

El generador de señal local está implementado mediante una memoria, lo que significa que las fases locales están definidas con una cierta resolución que depende del número de muestras por chip almacenadas en dicha memoria. Por este motivo la salida del filtro del bucle tiene que ser analizada por el bloque de lógica de control para corregir el retardo estimado siempre y cuando el error de estimación sea superior a la resolución de la fase de la réplica local. La fase local actualizada es finalmente introducida en el generador de señal local, cerrándose así el bucle de seguimiento.

La etapa de seguimiento ELDLL tiene dos compromisos de diseño principales: el espaciado *early-late* Δ entre las respectivas réplicas y el ancho de banda del filtro del bucle W_L. Como ya se mencionó

anteriormente, el perfil de la función del discriminador depende fuertemente del espaciado Δ, influyendo directamente en el efecto del ruido y del multicamino en el rendimiento del seguimiento. Por un lado, un espaciado estrecho proporciona un discriminador de elevado flanco en la región de trabajo, lo que a su vez proporciona una sensibilidad más alta, y por tanto un menor error debido al ruido. Además, como se vio en [2], reduce el efecto de las componentes de multicamino en la estimación. Por otro lado, el estrechamiento del espaciado conlleva también la reducción de la región de trabajo donde la información entregada por el discriminador es adecuada para el seguimiento, lo que aumenta la probabilidad de que el ruido cause la pérdida del enganche. Respecto al ancho de banda del filtro del bucle W_L, cuanto menor sea éste mayor es la capacidad de filtrado de ruido, incrementándose así la precisión en condiciones estáticas. Sin embargo, la capacidad del filtro para seguir variaciones de la estimación del retardo de la señal de entrada se reduce, dando lugar a mayores errores dinámicos.

2.3.2.3 Demodulación DSSS

La etapa de demodulación DSSS tiene como función demodular mediante DSSS la señal entrante. La estimación de retardo proporcionada por la etapa de seguimiento ELDLL es empleada para generar una réplica local de la secuencia PRN esperada en fase con la misma. La precisión con que la estimación del retardo es proporcionada influye directamente en la coherencia con que la componente directa de la señal de entrada es demodulada, y en consecuencia en la precisión alcanzable en la estimación de fase posterior.

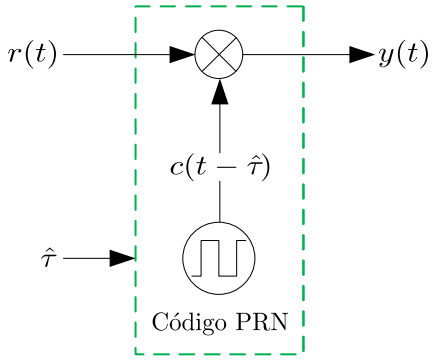


Figura 2.22: Diagrama de bloques de la etapa de demodulación DSSS.

La Figura 2.22 representa el esquema de demodulación DSSS. La secuencia PRN localmente generada $c(t-\hat{\tau})$ multiplica a la señal entrante $r(t)$ produciendo sobre ésta el desensanchamiento de su espectro y devolviéndola a su frecuencia original f_m. La demodulación de $r(t)$ da lugar a $y(t)$, cuya fase será estimada en la etapa siguiente. Cuanto más cerca esté el retardo estimado $\hat{\tau}$ por la etapa de seguimiento ELDLL del retardo de la componente LOS de $r(t)$, con mayor coherencia será demodulada, y por tanto mayor potencia de la misma se recuperará: $|\tau_{LOS} - \hat{\tau}| \approx 0 \rightarrow y(t) \approx r_{LOS}(t)$ en una situación libre de multicamino (2.20); del mismo modo, las componentes MP de $r(t)$ serán demoduladas con menor coherencia, y menor potencia de éstas se recuperará, por lo que la estimación de fase será más precisa.

$$r_{LOS}(t) = \sin(2\pi f_m t + \varphi_0 + \varphi_{LOS}) \tag{2.20}$$

2.3.2.4 Estimación de fase

La etapa de estimación de fase tiene como función proporcionar una estimación de la fase de la señal de entrada. La fase es estimada mediante dos procesos: el primero de ellos consiste en la demodulación I/Q asíncrona de la señal demodulada mediante DSSS; el segundo, es la extracción de la información de fase

mediante una estimación por arcotangente. La precisión de la estimación de fase depende fuertemente de la coherencia con que es demodulada mediante DSSS la señal entrante, ya que ésta define la potencia de cada componente, y por tanto de la estimación de retardo proporcionada por la etapa de seguimiento ELDLL.

La Figura 2.23 muestra el diagrama de bloques de la etapa de estimación de fase. La señal $y(t)$ es multiplicada por sus referencias en fase $I_{ref}(t)$ (2.21) y en cuadratura $Q_{ref}(t)$ (2.22). Las señales de referencia $I_{ref}(t)$ y $Q_{ref}(t)$ son generadas localmente a la frecuencia $f_m + \Delta f$ y desfasadas $\pi/2$ entre sí. Debido a que la señal emitida por el robot móvil y las señales de referencia I/Q son generadas por osciladores distintos la frecuencia de ambas señales no es idéntica, sino que existe una pequeña diferencia dependiente de la estabilidad en frecuencia ε_{freq} de cada oscilador y que es modelada mediante el parámetro Δf. Tras la multiplicación se obtienen las señales $y^I(t)$ (2.23) y $y^Q(t)$ (2.24), ambas compuestas por una componente de alta frecuencia $(2 \cdot f_m + \Delta f)$ y otra componente de muy baja frecuencia (Δf). A continuación las componentes de alta frecuencia de las señales $y^I(t)$ y $y^Q(t)$ son eliminadas mediante un filtro paso-bajo con un ancho de banda $W_{I/Q}$, obteniéndose las señales de baja frecuencia $y^{I'}(t)$ (2.25) y $y^{Q'}(t)$ (2.26). Finalmente la fase $\hat{\varphi}$ (2.27) es extraída mediante una estimación por arcotangente a partir de las señales $y^{I'}(t)$ y $y^{Q'}(t)$.

$$I_{ref}(t) = \sin(2\pi(f_m + \Delta f)t) \tag{2.21}$$

$$Q_{ref}(t) = \cos(2\pi(f_m + \Delta f)t) \tag{2.22}$$

$$y^I(t) = \cos(2\pi(2f_m + \Delta f)t + \varphi_0 + \varphi_{LOS}) + \cos(2\pi\Delta f t + \varphi_0 + \varphi_{LOS}) \tag{2.23}$$

$$y^Q(t) = \sin(2\pi(2f_m + \Delta f)t + \varphi_0 + \varphi_{LOS}) + \sin(2\pi\Delta f t + \varphi_0 + \varphi_{LOS}) \tag{2.24}$$

$$y^{I'}(t) = \cos(2\pi\Delta f t + \varphi_0 + \varphi_{LOS}) \tag{2.25}$$

$$y^{Q'}(t) = \sin(2\pi\Delta f t + \varphi_0 + \varphi_{LOS}) \tag{2.26}$$

$$\hat{\varphi} = \arctan\left(\frac{y^{Q'}(t)}{y^{I'}(t)}\right) = 2\pi\Delta f t + \varphi_0 + \varphi_{LOS} \tag{2.27}$$

Como se puede observar, la fase estimada $\hat{\varphi}$ está compuesta por el término $2\pi\Delta f t$, por la fase inicial φ_0 con que el robot móvil emite y cuyo valor es desconocido, y por la fase añadida φ_{LOS} debido al camino directo que recorre la señal entre emisor y receptor. La estimación diferencial de fase obtenida al restar la fase estimada en un receptor con la fase estimada en otro permite eliminar las componentes $2\pi\Delta f t$ y φ_0 ya que ambas son comunes a todos los receptores. Del mismo modo, la estimación diferencial de fase permite eliminar cualquier error de offset siempre y cuando sea común y simultáneo a todas las estimaciones.

Sobre la diferencia de fase final se realiza un último filtrado paso-bajo con el objetivo de limitar el ancho de banda final y así reducir la influencia del ruido todo lo posible. La función de transferencia de dicho filtro representa el principal compromiso de diseño de esta etapa. Un filtrado fuerte y abrupto reduce

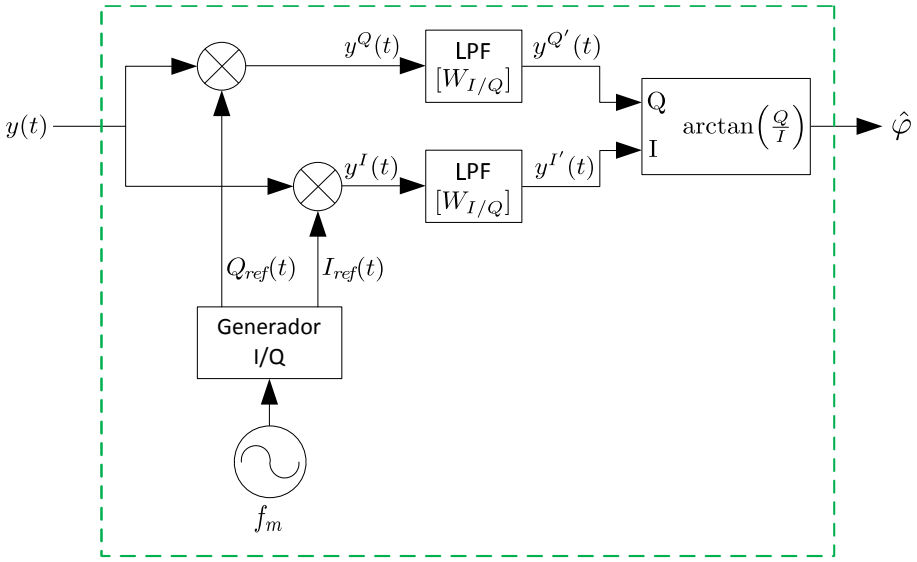


Figura 2.23: Diagrama de bloques de la etapa de estimación de fase.

la dispersión en las medidas estáticas mejorando su precisión; sin embargo, ello conlleva el incremento del error en las medidas dinámicas. El filtro debe permitir seguir las variaciones del movimiento del robot móvil sin acumular excesivo error.

Capítulo 3

Implementación hardware del sistema de medida

En este capítulo se describe la propuesta de implementación hardware para la arquitectura de medida diferencial de distancias descrita en el apartado 2.3.2. En primer lugar, se analizan los requisitos funcionales de la arquitectura, así como el soporte hardware sobre el que se implementará la misma (sección 3.1). En segundo lugar se describe detalladamente cómo se ha implementado la arquitectura así como la funcionalidad de cada uno de los bloques en los que se basa (sección 3.2).

3.1 Análisis previo

3.1.1 Requerimientos funcionales

La precisión alcanzable en la estimación diferencial de distancia final depende de los parámetros de diseño (frecuencia de modulación, frecuencia de chip, y relación de fase entre ambas frecuencias), y de la potencia óptica de los dispositivos fotónicos. La precisión global del sistema puede ser desglosada en dos partes: precisión en la estimación del retardo $\hat{\tau}$ proporcionado por la etapa de tracking, y precisión en la estimación de fase φ_{LOS}.

La precisión en la estimación del retardo depende directamente de la frecuencia de chip, de la potencia óptica con que la señal $r(t)$ alcanza cada receptor, y del ruido añadido en la etapa de acondicionamiento del receptor. A su vez, la precisión del retardo estimado influye en la coherencia con que la señal $r(t)$ es demodulada mediante DSSS, ya que cuanto más precisa sea la estimación más potencia se recupera, y en consecuencia, mayor es el nivel de SNR obtenido (SNR_{DSSS}).

La precisión en la estimación de fase depende en primer lugar de la raíz cuadrada del SNR_{DSSS}, y en segundo lugar de la frecuencia de modulación. Por un lado, el nivel de SNR tras la demodulación DSSS afecta a la precisión de la estimación de fase en sí misma; por otro lado, como ya se vio en las ecuaciones (2.1) y (2.2), la sensibilidad distancia-fase depende inversamente de la frecuencia de modulación, de forma que cuanto mayor es ésta, menor influencia tiene el error cometido en la estimación de fase en la estimación de distancia.

Como ya se comentó en el apartado 2.3.2, los valores de los parámetros de diseño deben maximizar la precisión global del sistema respetando el ancho de banda W_{IR} del enlace IR. Para ello, las frecuencias de modulación y de chip así como la potencia óptica deben ser lo más elevadas posibles. Sin embargo,

en los dispositivos fotónicos actuales existe un compromiso entre la frecuencia de trabajo y la potencia óptica. En los emisores, como se muestra en la Figura 3.1, se busca trabajar a la frecuencia que maximice el producto frecuencia por potencia, frecuencia que en los últimos años se ha incrementado, y que hoy en día se sitúa aproximadamente en torno 50 MHz. Por debajo de esta frecuencia pueden encontrarse dispositivos más lentos pero que mantienen la misma potencia, y por encima los dispositivos son más rápidos pero su potencia cae también rápidamente. En los receptores se busca aumentar el área activa para aumentar la potencia captada, aumento que conlleva un incremento de la capacidad parásita, y por consiguiente, una reducción del ancho de banda.

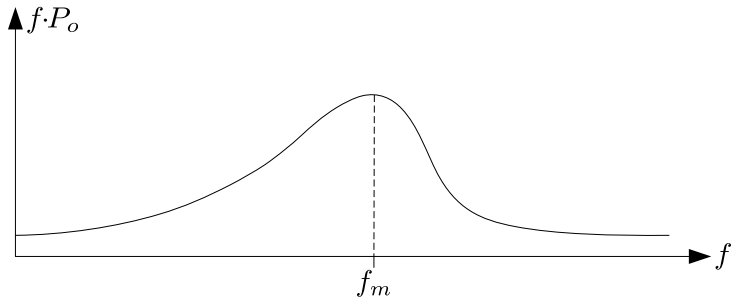


Figura 3.1: Relación entre la frecuencia y potencia de un emisor fotónico.

Teniendo en cuenta estas consideraciones, se ha definido $f_m = f_{chip} = 25$ MHz asumiendo un enlace con $W_{IR} = 50$ MHz. El motivo por el que ambas frecuencias son iguales es que de esta forma cada chip de la secuencia PRN ensanchadora contiene un seno de la señal AM sinusoidal. El valor de ambas frecuencias se ha establecido en $W_{IR}/2$ de forma que el lóbulo principal de la señal *sinc* que conforma su espectro se sitúe centrado en el ancho de banda del enlace debido a que este lóbulo contiene más del 90 % de la potencia total de la señal. Dada $f_m = 25$ MHz, la sensibilidad distancia-fase de la arquitectura de medida es $K_{d/\varphi} = 3.33$ cm/$^{\text{o}}$ (veánse las ecuaciones (2.1) y (2.2)). Por otra parte, para $f_m = 25$ MHz la frecuencia mínima de muestreo del sistema debe ser $f_s = 50$ MHz (Nyquist). Cuanto mayor sea la frecuencia de muestreo mayor será el número de muestras/chip, y en consecuencia más precisa será la estimación del retardo.

En cuanto a la relación de fase φ_0 entre la señal AM y la secuencia PRN, dado que debe minimizar la distorsión durante las transiciones de fase, y que cada chip contiene un seno ($f_{chip} = f_m$), se ha establecido $\varphi_0 = 0^\circ$ para que las transiciones se realicen en los cruces por cero del seno, reduciendo así el efecto de la limitación del ancho de banda.

3.1.2 Soporte de implementación

Para realizar la implementación hardware de la arquitectura de medida diferencial de distancias se han barajado los siguientes dispositivos hardware: micro-controlador (μC), ASIC *(Application-Specific Integrated Circuit)*, y FPGA *(Field Programmable Gate Array)*. En la Tabla 3.1 se muestra una comparativa entre los distintos dispositivos respecto a su rendimiento y coste.

- Micro-controlador: sistema digital programable de propósito general que proporciona al diseñador una gran flexibilidad para poder realizar diversas aplicaciones. Ello permite que los μC's puedan ser fabricados masivamente reduciéndose así sus costes de fabricación. Sin embargo, al tratarse de un dispositivo válido para múltiples aplicaciones, para una aplicación concreta sus características hardware no están optimizadas, lo que le impide alcanzar un elevado rendimiento en cuanto a frecuencias de funcionamiento.

- ASIC: circuito integrado diseñado *ad hoc* para una aplicación específica, lo que por un lado incrementa drásticamente sus costes de fabricación tanto temporales como económicos, pero por otro lado, sus características hardware están optimizadas para dicha aplicación, proporcionando un rendimiento superior al de cualquier otro dispositivo hardware.
- FPGA: a diferencia de un ASIC dispone de recursos lógicos y de interconexión que son configurables por el diseñador una vez que ésta ya ha sido fabricada, lo que le dota de cierta flexibilidad y al mismo tiempo reduce su rendimiento en comparación a un ASIC; en consecuencia, sus costes de fabricación son menores. El mayor grado de flexibilidad de las FPGA's permite que puedan ser empleadas en distintos tipos de aplicaciones, similares pero menos exigentes, que en las que se emplean ASIC's, aunque con un mayor consumo de potencia. Su utilización resulta adecuada para diseños pequeños y volúmenes de producción bajos.

Tabla 3.1: Comparativa rendimiento-coste de los distintos dispositivos HW.

Dispositivo	**Rendimiento**	**Coste**
μC	Bajo	Bajo
ASIC	Alto	Alto
FPGA	Medio	Medio

La elección del dispositivo hardware no depende exclusivamente de los requerimientos funcionales del sistema, sino que también debe tenerse en cuenta que éste a su entrada debe llevar conectada una etapa de digitalización y acondicionamiento de señal que suministre la información a la arquitectura de medida. Esta etapa estará basada principalmente en un SAD de altas prestaciones: elevada frecuencia de muestreo, multicanal, y de adquisición simultánea. El dispositivo escogido debe ser compatible con esta etapa principalmente en cuanto a frecuencias se refiere.

En vista de los requerimientos funcionales vistos anteriormente, el micro-controlador, a pesar de tratarse de la opción más económica, resulta inviable debido a que, en combinación con la etapa de digitalización, su rendimiento no alcanza el que el sistema requiere ($f_{CLK} = f_s >> 2 \cdot f_m$). Por otro lado, el ASIC proporciona un rendimiento insuperable por los otros dos dispositivos, pero su elevado coste económico obliga a tener que descartarlo. En cambio, la FPGA constituye una solución intermedia entre el micro-controlador y el ASIC en lo que a rendimiento y coste se refiere. Finalmente, tras analizar detenidamente el rendimiento y coste de cada dispositivo, se ha decidido que una FPGA es el dispositivo más adecuado sobre el cual llevar a cabo la implementación del sistema dado que el rendimiento que proporciona cumple con los requerimientos funcionales de la arquitectura y su coste económico resulta asequible. De los numerosos modelos de FPGA disponibles en el mercado se ha seleccionado el modelo Spartan-6 XC6SLX45T-FGG484 -3C debido a que sus características hardware y recursos lógicos cubren con suficiencia las exigencias funcionales del sistema (permite alcanzar una frecuencia de funcionamiento de hasta 1080 MHz). La FPGA elegida, perteneciente a la serie 6 de la familia Spartan de Xilinx, se muestra en la Figura 3.2 integrada en la tarjeta de evaluación SP605. Esta tarjeta dispone, entre otros muchos componentes, de un oscilador de 200 MHz y además incluye un puerto de expansión con un conector FMC - LPC (parte superior izquierda) que permite conectarle un módulo de digitalización externo. El módulo escogido, mostrado en la Figura 3.3, es el FMC AD-FMCOMMS1-EBZ de Analog Devices, y consta de un ADC AD9643 y un DAC AD9122. El AD9643 es un ADC de 14 bits, multicanal dual, de adquisición simultánea, y que alcanza velocidades de muestreo de hasta 250 MS/s.

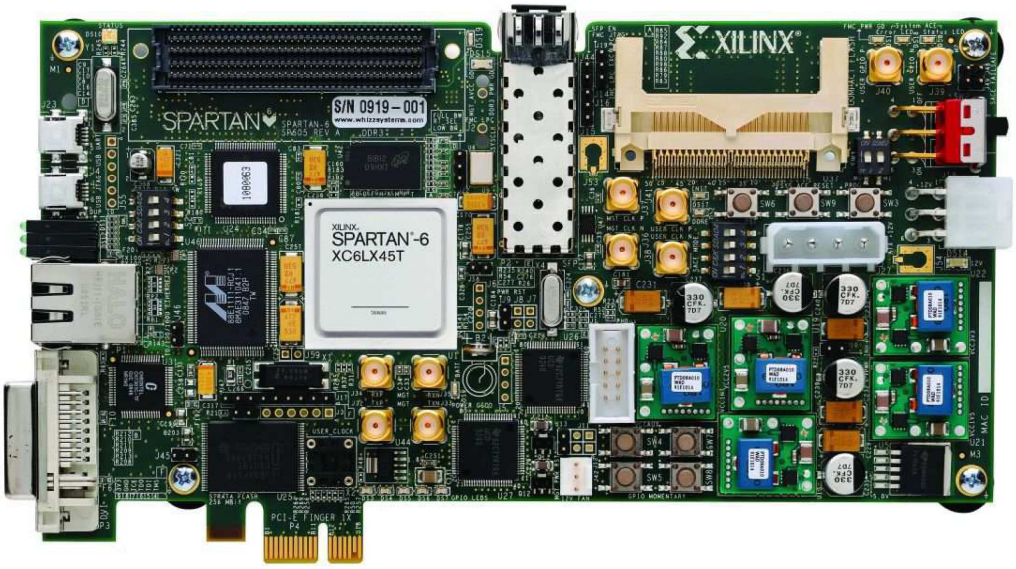


Figura 3.2: Tarjeta de evaluación SP605

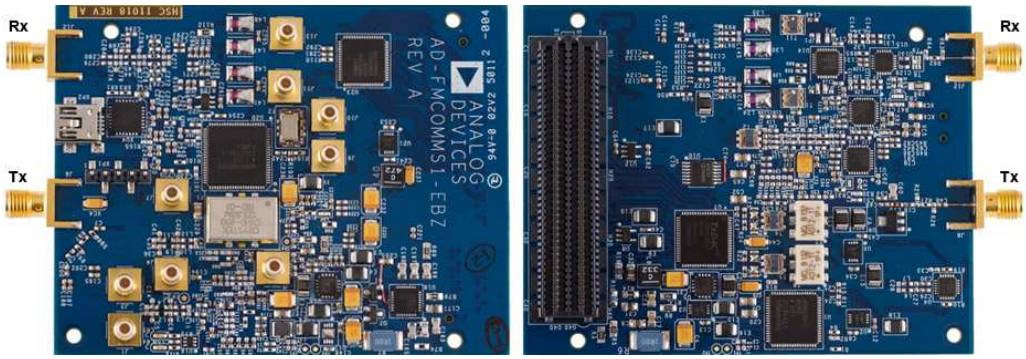


Figura 3.3: Módulo de digitalización FMC AD-FMCOMMS1-EBZ

3.2 Desarrollo del sistema

La implementación realizada del sistema de medida diferencial de distancias proporciona la estimación de distancia diferencial entre el robot móvil y dos receptores cualesquiera del entorno, siendo extrapolable a cualquier otra pareja de receptores. Los recursos lógicos y de interconexión de la FPGA han sido configurados mediante el lenguaje de descripción hardware VHDL. La metodología de diseño seguida, así como la estructura de esta sección, es top-down. Los componentes que conforman el diseño son referidos indistintamente también como módulos, cores, o IP's *(Intellectual Properties)*. Para ilustrar con mayor claridad el funcionamiento de cada una de las etapas del sistema se adjunta una simulación funcional de las mismas en sus respectivos apartados.

Las señales de entrada a la FPGA son dos tonos sinusoidales r(t) con una frecuencia de modulación de 25 MHz, una fase inicial nula, y ensanchados por una secuencia PRN de 6 bits. Ambas señales, procedentes de dos de los receptores situados en el plano superior de la CBL, han sido digitalizadas previamente por un ADC de 8 bits ($bits_{ADC}$), multicanal, y de adquisición simultánea (condición imprescindible). El ADC proporciona las señales digitalizadas en coma fija con formato $Q_{ADC} = [8, 7]$ (palabra de 8 bits con 7 bits de parte fraccionaria) en complemento a 2.

Los bloques de memoria RAM (BRAM) disponibles en la FPGA en los cuales se almacenan las réplicas de la señal emitida, la secuencia PRN para realizar la demodulación DSSS, y las señales de referencia I/Q, presentan un retardo lógico intrínseco de aproximadamente 7 ns. Considerando además los retardos lógicos y de interconexión entre el resto de los diversos recursos lógicos implicados, el periodo mínimo de funcionamiento alcanzado es de 9.858 ns. Por ello, las frecuencias de funcionamiento de la FPGA y de muestreo del ADC se han fijado en 100 MHz, de forma que cada muestra de señal proporcionada por el ADC se procese en un ciclo de reloj de la FPGA y no se pierda información.

La secuencia PRN es generada mediante un LFSR *(Linear Feedback Shift Register)*. La generación de una época de la secuencia PRN depende los siguientes parámetros: número de bits del LFSR ($bits_{PRN}$) y frecuencia de chip (f_{chip}). El número de bits con que se genera la secuencia PRN únicamente afecta al número de chips del que consta la secuencia, puesto que el ancho de banda de correlación está limitado por el filtro del bucle de la etapa de tracking ELDLL. Por cuestiones de sencillez, se ha optado por un LFSR de 6 bits. Dado un LFSR con $bits_{PRN} = 6$ y $f_{chip} = 25$ MHz, la secuencia PRN generada presenta una longitud $N_{chip} = 63$ chips y una duración $t_{PRN} = 2.52$ μs (3.1). La duración de la secuencia PRN determina la duración de una época de las señales de entrada (t_{signal}), y además establece el periodo de correlación (T_f).

$$t_{PRN} = \frac{N_{chip}}{f_{chip}} \qquad N_{chip} = 2^{bits_{PRN}} - 1 \tag{3.1}$$

El número de muestras en una época de señal está definido por el producto del número de chips de la secuencia PRN por el número de muestras por chip: $n_{samples} = N_{chip} \cdot samples/chip$. El número de muestras por chip viene determinado por el cociente f_s/f_{chip}. Para $f_s = 100$ MHz y $f_{chip} = 25$ MHz se tienen 4 muestras/chip; como la secuencia PRN generada tiene $N_{chip} = 63$ chips, el número de muestras en cada época de señal es de $n_{samples} = 63 \cdot 4 = 252$ muestras. El retardo máximo de las señales de entrada puede ser de hasta 252 muestras, por lo que para codificarlo en binario son necesarios 8 bits ($bits_{samples}$).

La implementación del sistema de medida se ha realizado en el componente DDMS *(Differential Distance Measuring System)*, y su función es proporcionar la estimación de distancia diferencial entre el robot móvil y dos receptores a partir de las señales de entrada Signal I y Signal J. El componente DDMS está compuesto por el core DCM *(Digital Clock Manager)*, dos módulos PMS *(Phase Measuring*

System), el módulo Differential Phase, y el módulo Phase - Distance. La Figura 3.4 representa el diagrama de bloques del diseño realizado. Las señales de entrada y salida de la FPGA están en coma fija, en complemento a 2, y en formato $Q_{ADC} = [8, 7]$ Signal I y Signal J, y en formato $Q_{DIST} = [36, 31]$ ($bits_{DIST} = 36$) Differential Distance.

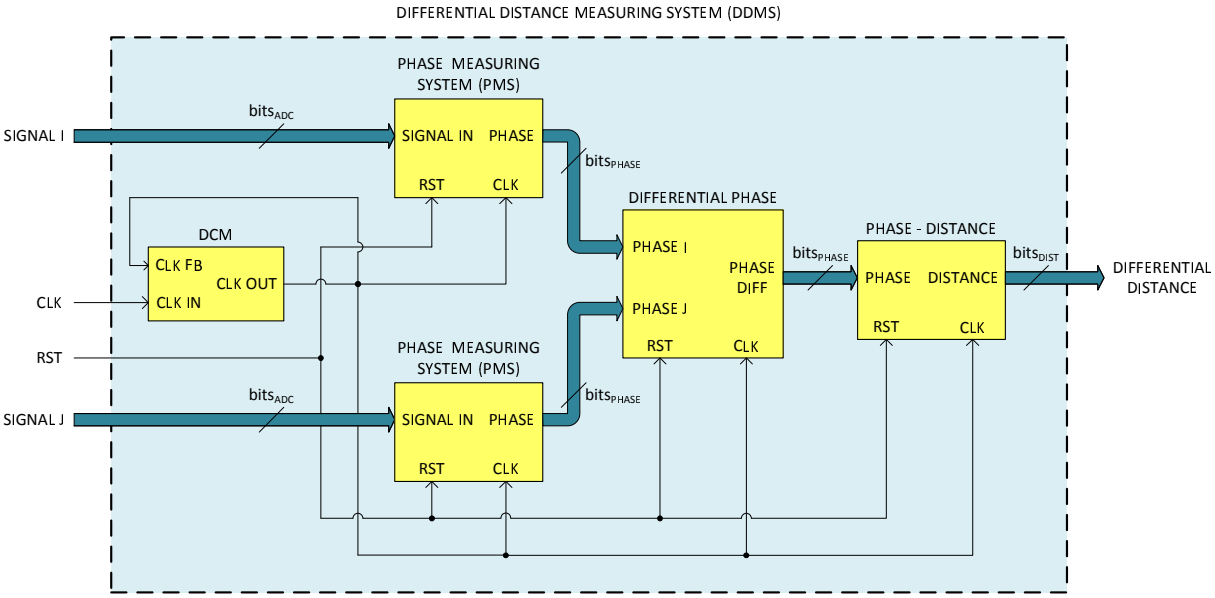


Figura 3.4: Diagrama de bloques VHDL del sistema de medida de distancia diferencial.

El DCM es un gestor de reloj digital que permite implementar múltiples funciones, entre las que destacan: bucle enganchado en retardo (DLL), sintetizador de frecuencia digital (DFS), y desplazador de fase digital (DPS). Las rutas de interconexión de la FPGA no son ideales, sino que presentan retardos temporales que provocan que los flancos de la señal de reloj (CLK) no lleguen simultáneamente a todos los componentes, lo que puede llegar a causar la pérdida de sincronismo del diseño. El *skew* es el máximo retardo de la señal de reloj entre dos recursos lógicos adyacentes del sistema. El core DCM diseñado, basado en la primitiva *DCM_SP*, implementa un DLL que se emplea para eliminar el *skew* de la señal de reloj y garantizar así el sincronismo del sistema. El *skew* es eliminado alineando la salida de reloj generada por el DCM (CLK OUT) con otra versión de la misma que es realimentada (CLK FB).

Cada uno de los módulos PMS se encarga de proporcionar la estimación de la fase $\hat{\varphi}$ en radianes de la señal que entra en él en formato $Q_{PHASE} = [16, 13]$ ($bits_{PHASE} = 16$). En la Figura 3.5 se muestra el diagrama de bloques de la arquitectura interna de un módulo PMS, el cual está formado por cuatro IP's: Synchronization C/A, Tracking ELDLL, Demodulation DSSS, y Phase Estimation. Cada uno de estos IP's implementa la función que realizan las distintas etapas que conforman el sistema de medida diferencial de distancias, y que en los siguientes apartados de esta sección se explicará cómo se ha implementado su funcionalidad.

El módulo Differential Phase obtiene a partir de las estimaciones de fase generadas por los módulos PMS la estimación de fase diferencial $\hat{\varphi}_{ij}$. Dado que $\hat{\varphi}_i$ y $\hat{\varphi}_j$ son proporcionadas dentro del intervalo $\pm$ π rad, para garantizar que la estimación diferencial de fase se encuentra dentro de dicho intervalo, ésta debe ser corregida en dos situaciones:

- Si $\hat{\varphi}_i - \hat{\varphi}_j > \pi$ rad $\rightarrow \hat{\varphi}_{ij} = \hat{\varphi}_i - \hat{\varphi}_j$ - 2π rad
- Si $\hat{\varphi}_i - \hat{\varphi}_j < -\pi$ rad $\rightarrow \hat{\varphi}_{ij} = \hat{\varphi}_i - \hat{\varphi}_j + 2\pi$ rad

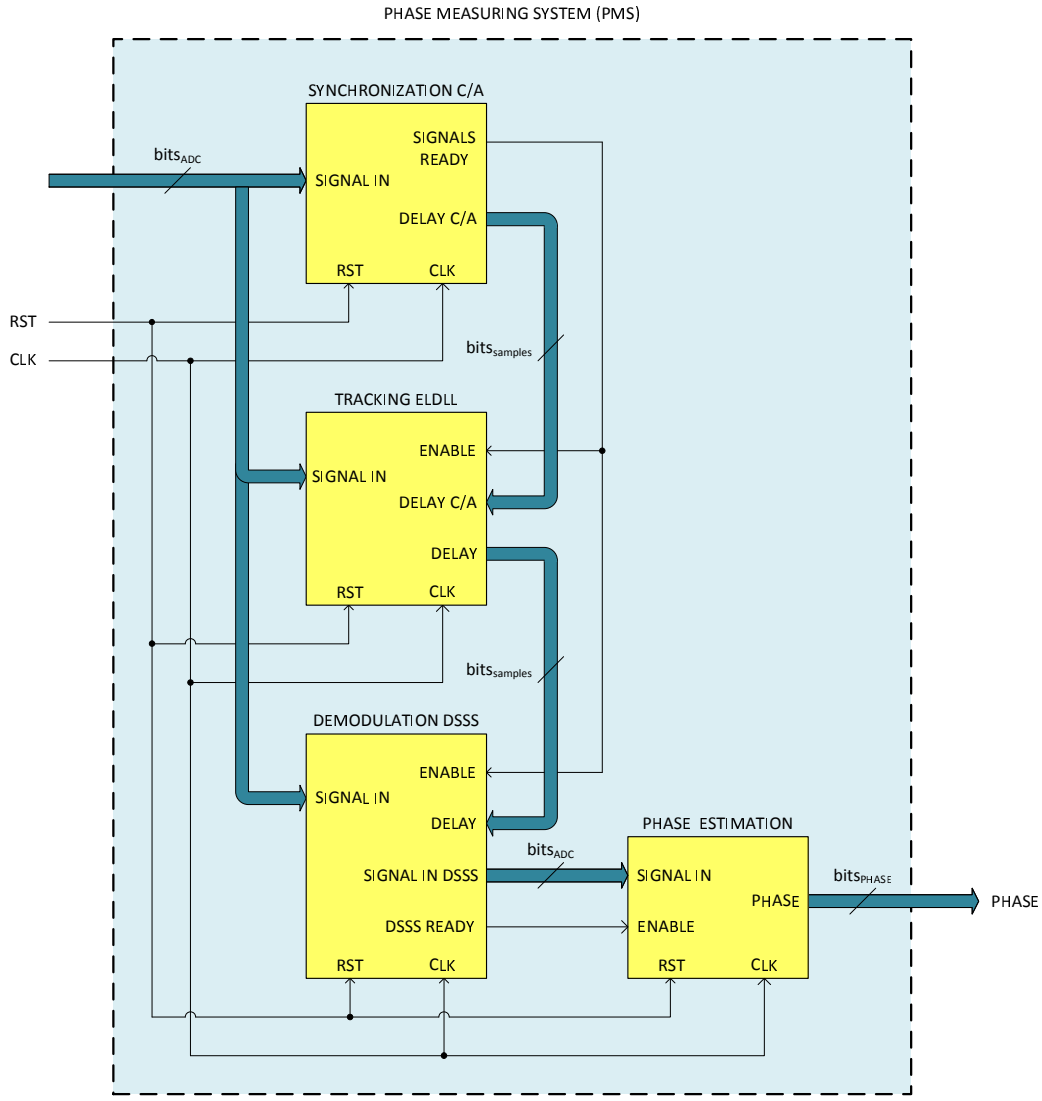


Figura 3.5: Diagrama de bloques VHDL del sistema de medida de fase.

Cuando la diferencia de fase entre las señales de entrada se encuentra muy próxima a $\pm\ \pi$ rad, puede suceder que la estimación varíe brusca y continuadamente entre $\pm \sim \pi$ rad (principalmente cuando el SNR de las señales es bajo) obteniéndose una estimación errónea. Para solventar este inconveniente, el módulo Differential Phase dispone del componente Phase Correction. Este componente compara la estimación de fase diferencial actual con la anterior, y en caso de que la diferencia supere un umbral cercano a $\pm \sim 2\pi$ rad, la estimación actual es corregida tomando el valor de la estimación anterior, eliminándose de este modo las variaciones bruscas.

La estimación diferencial de fase es convertida a distancia mediante el módulo Phase - Distance. Este módulo se basa en el core Multiplier p2d que multiplica la estimación de fase (Q_{PHASE}) por la sensibilidad $K_{d/\varphi}$. Debe tenerse en cuenta que la estimación de fase es proporcionada en radianes, por lo que la sensibilidad $K_{d/\varphi}$ en [m/º] determinada por las ecuaciones (2.1) y (2.2) debe convertirse a [m/rad] tal como se indica en la ecuación (3.2). El nuevo valor de sensibilidad obtenido se ha cuantificado en formato $Q_{d/\varphi} = [20,\ 18]$ para no perder precisión en el proceso de cuantificación. Finalmente, la estimación diferencial de distancia es proporcionada en metros en formato $Q_{DIST} = [36,\ 31]$.

$$K_{d/\varphi}[m/rad] = K_{d/\varphi}[m/^{\circ}] \cdot \frac{360^{\circ}}{2\pi\ rad} = \frac{1}{30} \cdot \frac{360}{2\pi} = \frac{6}{\pi} \tag{3.2}$$

3.2.1 Sincronización C/A

El objetivo del IP Synchronization C/A es proporcionar la estimación gruesa del retardo (Delay C/A) de la señal entrante (Signal In) con una precisión de ± 1.0 chip. En la Figura 3.6 se puede ver el diagrama de bloques del IP Synchronization C/A, el cual está compuesto por los módulos Correlator C/A, Buffer C/A, y Peak Detection.

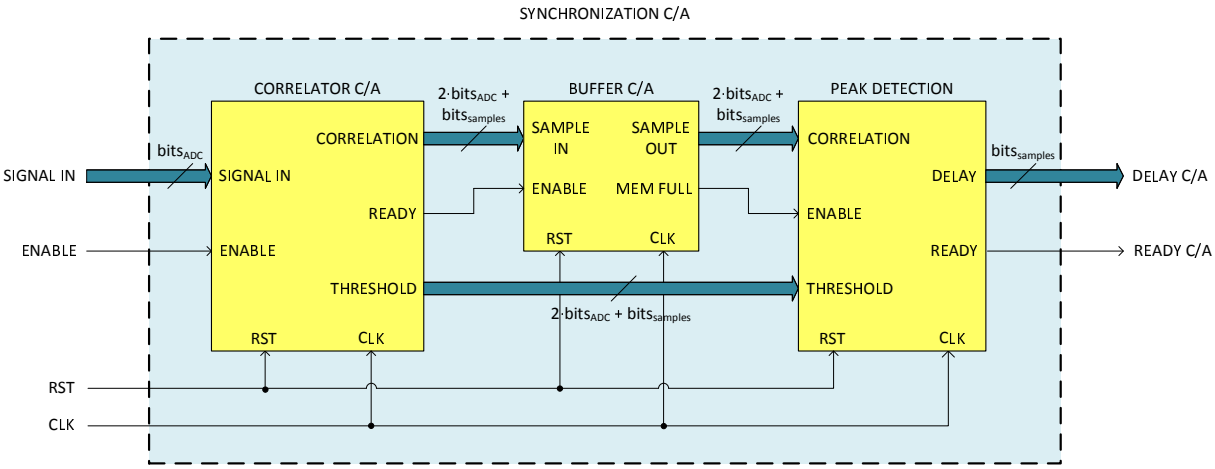


Figura 3.6: Diagrama de bloques VHDL de la etapa de sincronización C/A.

El módulo Correlator C/A implementa un correlador que se encarga de calcular los valores de correlación (Correlation) entre la señal entrante (Signal In) y la réplica local de la señal esperada; además calcula el umbral (Threshold) empleado en la detección de pico. El correlador C/A está basado en el componente Buffer Mem C/A y en la macro MACC *(Multiply ACCumulate)* de la primitiva *DSP48A1*. El buffer consiste en el core Memory Local, que implementa una memoria local basada en la primitiva *RAMB16BWER*, y que contiene una réplica de una época de la señal emitida. El MACC calcula la correlación entre una época de la señal de entrada y la réplica de la señal esperada generada localmente por el componente Buffer Mem C/A. En cada periodo de correlación la réplica local es generada con una fase diferente con una resolución $\Delta = f_{chip}/f_s = 0.25$ chips. Puesto que una época de señal contiene $n_{samples}$ muestras, para barrer todas las posibles fases de ésta son necesarios $n_{samples}$ periodos de correlación. Debido a que la memoria que contiene la réplica local de la señal esperada presenta dos ciclos de latencia, la señal de entrada ha sido retardada dos ciclos de reloj para que en el momento de calcular las correlaciones ambas señales estén sincronizadas. La Figura 3.7 muestra el diagrama de bloques del módulo Correlator C/A.

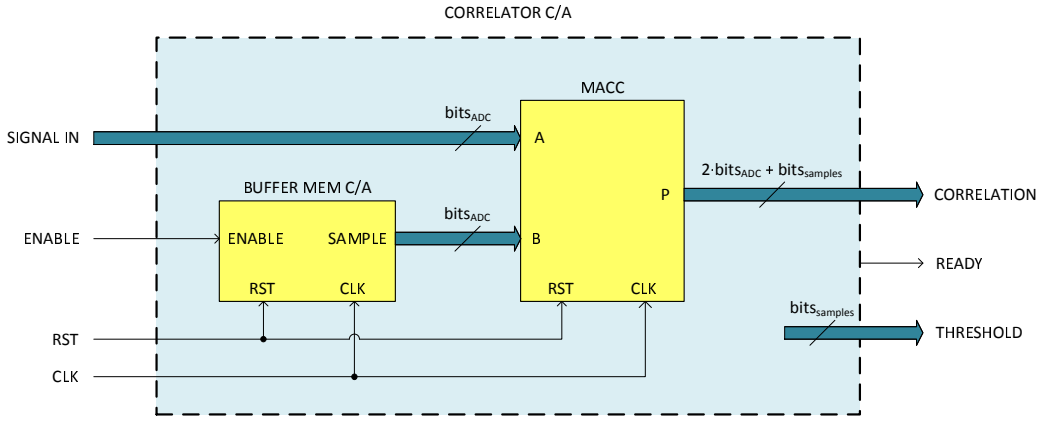


Figura 3.7: Diagrama de bloques VHDL del correlador C/A.

El umbral empleado en la detección de pico se ha establecido como el 75 % del valor máximo de correlación ($R_{m,máx}$) de manera que no se vea afectado por el ruido que presente la señal de entrada. En la Figura 3.8 se muestra la función de correlación, tanto continua como muestreada, centrada en la región del pico de correlación entre la señal de entrada y la réplica local en función de la diferencia de sus respectivos retardos τ_{LOS} y τ_L. La obtención del máximo de la función de correlación se realiza dinámicamente a partir de los sucesivos valores de correlación que se van obteniendo.

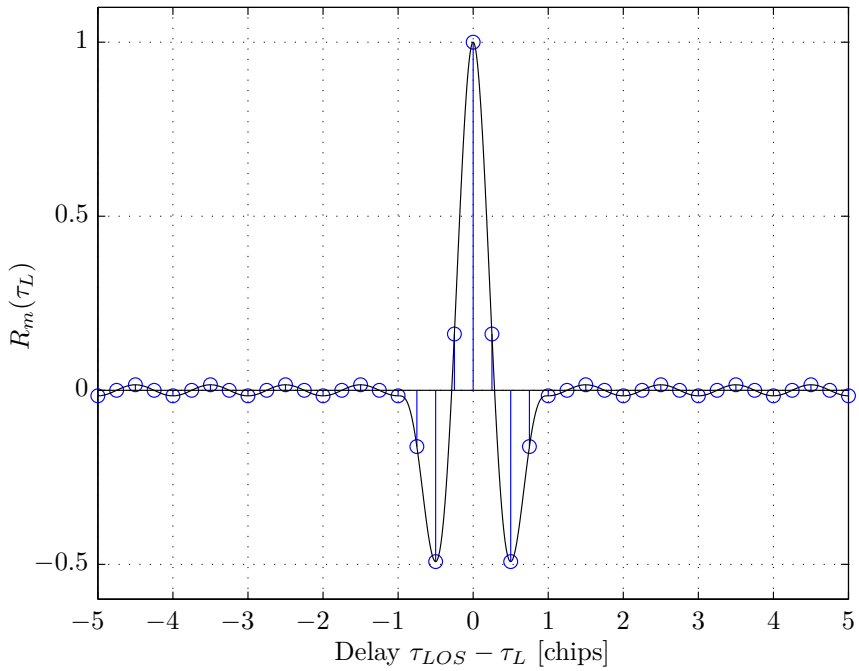


Figura 3.8: Función de correlación en función del retardo entre la señal de entrada y local.

Cada vez que una correlación es calculada, indicado por la señal Correlation Ready, su valor es almacenado en el módulo Buffer C/A. Este buffer se basa en el core Memory C/A, que también implementa una memoria local basada en la primitiva *RAMB16BWER*, y que tiene una doble función: almacenar los valores de correlación correspondientes a todas las posibles fases de la réplica local, y una vez almacenados entregarlos al módulo Peak Detection. Este módulo compara dichos valores de correlación con el umbral (Threshold) previamente calculado por el módulo Correlator C/A. Cuando un valor de correlación supera el umbral, la fase de la réplica local asociada a dicho valor se establece como la estimación gruesa del retardo (Delay C/A), y se activa a nivel alto la señal Ready C/A, la cual habilita el funcionamiento de los IP's Tracking ELDLL y Demodulation DSSS. La precisión de la estimación gruesa del retardo debe estar comprendida entre ± 1.0 chip, de forma que si el valor de correlación que supera el umbral no es exactamente el del pico sino uno próximo a él, el retardo pueda ser corregido posteriormente en la etapa de tracking.

La Figura 3.9 muestra el cronograma obtenido al realizar una simulación funcional en la que la señal de entrada presenta una fase inicial $\varphi_0 = 0^o$ y un retardo $\tau_{LOS} = 131$ muestras. Se observa que la señal Correlation C/A presenta un pico claramente diferenciado, indicando que la señal de entrada y la réplica local están en fase, y que su perfil es prácticamente igual al de la función de correlación de la Figura 3.8. Además pueden verse pulsos de alta frecuencia en la señal Correlation Ready que indican que se ha realizado la correlación entre la señal de entrada y la réplica local. Una vez barridas todas las posibles fases de la réplica local, se obtiene el valor del umbral Threshold C/A, y se activa a nivel alto la señal Mem Full

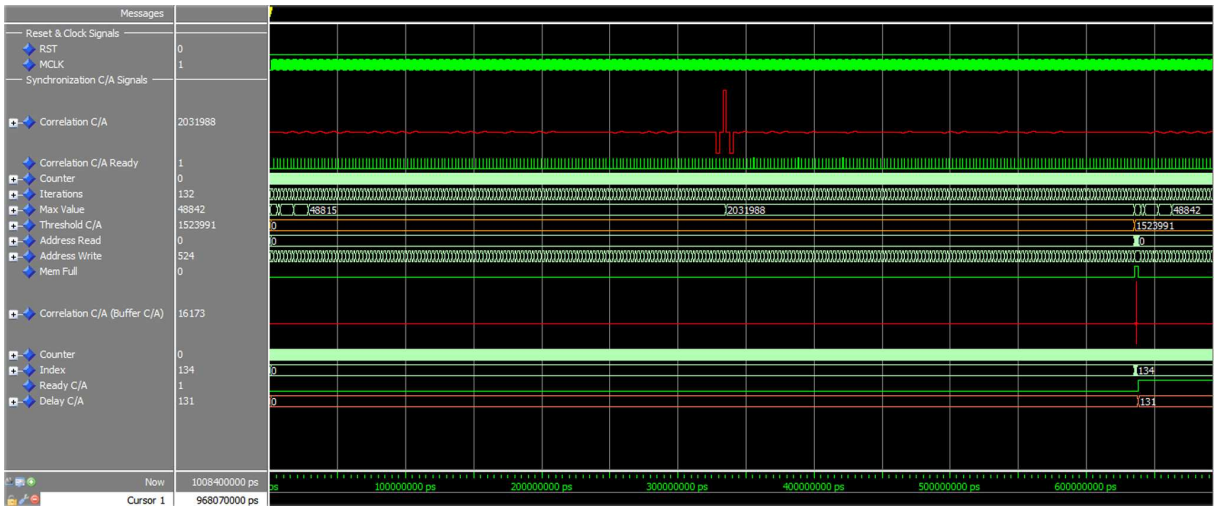


Figura 3.9: Cronograma funcional de la etapa de sincronización C/A.

únicamente durante el siguiente periodo de correlación. En dicho periodo se habilita el funcionamiento del módulo Peak Detection, el cual recibe el umbral de detección junto con los valores de correlación almacenados anteriormente en el módulo Buffer C/A (Correlation C/A Buffer); en base a ello, proporciona la estimación gruesa (Delay C/A) del retardo y activa la señal Ready C/A, dando por concluida esta etapa. El umbral Threshold C/A = 1523991 obtenido corresponde al 75 % del valor máximo de correlación $R_{m,máx} = 2031988$. La estimación gruesa del retardo generada es de 131 muestras, coincidiendo con el retardo de la señal de entrada ($\hat{\tau}_0 = \tau_{LOS}$), por lo que el IP Synchronization C/A funciona correctamente. La etapa de sincronización C/A tiene una duración: $t_{sync_{C/A}} = (n_{samples} + 1) \cdot T_f = (252 + 1) \cdot 2.52\ \mu s = 637.56\ \mu$s (requiere un periodo de correlación adicional para realizar el proceso de detección de pico), y además, introduce dos ciclos de latencia en el sistema que deben ser tenidos en cuenta en las siguientes etapas para mantener el sincronismo del mismo.

3.2.2 Tracking ELDLL

El IP Tracking ELDLL se encarga de proporcionar la estimación continua del retardo (Delay) de la señal de entrada (Signal In) a partir de la estimación gruesa del retardo (Delay C/A) generada por el IP Synchronization C/A una vez que ésta se encuentra disponible (Enable). La Figura 3.10 representa el diagrama de bloques del IP Tracking ELDLL. El bucle de seguimiento ELDLL se ha implementado a través de los componentes Correlator Early-Late y Logic Control.

El componente Correlator Early-Late implementa un correlador *early-late*, y está compuesto por los módulos Correlator Early y Correlator Late, y por el módulo Delay Estimation. Los módulos Correlator Early y Correlator Late generan los valores de las correlaciones *early* y *late* (Correlation Early y Correlation Late) así como una señal que indica cuándo ambos valores están disponibles (Ready E-L). El módulo Delay Estimation tiene como función proporcionar la estimación continua del retardo (Delay). La Figura 3.11 muestra el diagrama de bloques de la estructura del componente Correlator Early-Late.

Los módulos Correlator *Early* y *Late* calculan la correlación entre la señal de entrada (Signal In) y la réplica local de ésta en fase adelantada (*early*) y retrasada (*late*) una muestra respectivamente. Ambos correladores están formados por el componente Buffer Mem E/L y la macro MACC. Este buffer, basado también en el core Memory Local, es el encargado de generar la réplica local en fase de la señal esperada con una muestra de adelanto o atraso según el correlador al que corresponde. Para generar las réplicas locales *Early* y *Late* el buffer toma la estimación continua del retardo (Delay) procedente del módulo

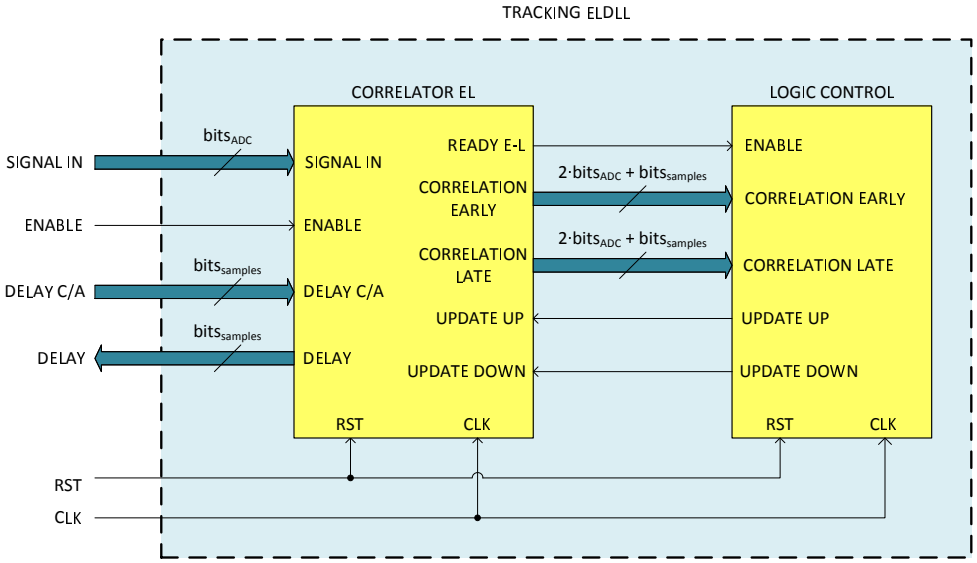


Figura 3.10: Diagrama de bloques VHDL de la etapa de tracking ELDLL.

Delay Estimation, le resta o suma una muestra respectivamente, y a partir de esa dirección extrae el contenido de la memoria local. El espaciamiento entre las réplicas *early* y *late* respecto a la réplica en fase es $\Delta = f_{chip}/f_s = 0.25$ chips. En la Figura 3.12 se representa el esquema de los correladores *Early* y *Late*.

Para garantizar que los valores de correlación suministrados a la lógica de control son los correctos, por un lado la señal Ready E-L genera un pulso únicamente cuando se producen pulsos simultáneamente en las señales Ready de los módulos Correlator Early y Late. Por otro lado la señal de entrada debe ser retardada cuatro ciclos para que esté sincronizada con las réplicas *early* y *late*, ya que las memorias de las que éstas proceden tienen también dos ciclos de latencia a los que hay que añadir los dos ciclos de latencia introducidos en el sistema por el IP Synchronization C/A.

El módulo Delay Estimation toma como entradas la estimación gruesa del retardo (Delay C/A) y las señales procedentes de la lógica de control Update Up y Update Down. Al inicio de cada periodo de correlación se comprueba si se ha recibido un pulso de alguna de las dos señales de control para actualizar la estimación del retardo (Delay). Cuando se detecta un pulso de la señal Update Up significa que el retardo estimado debe ser retrasado una muestra, mientras que un pulso de la señal Update Down implica adelantar una muestra el retardo estimado. La estimación del retardo es entregada a los correladores *early* y *late* para calcular sus respectivas correlaciones, y también es entregada a la etapa de demodulación DSSS.

La lógica de control, implementada por el componente Logic Control, toma los valores de las correlaciones *early* y *late* generados por el componente Correlator Early-Late. Cada vez que ambos valores están disponibles, indicado por un pulso en la señal Ready E-L, se calcula su diferencia (*early* - *late*) para obtener la función del discriminador. Esta función es filtrada a través del filtro del bucle, y su salida es posteriormente comparada respecto al umbral ± Threshold E/L, de forma que si lo supera por arriba se genera un pulso de la señal Update Up, y si lo supera por abajo de la señal Update Down. El umbral Threshold E/L se ha obtenido experimentalmente durante las pruebas de validación en las condiciones de ruido e interferencias por multicamino más desfavorables estableciéndose en $5 \cdot 10^5$. En este caso el umbral es estático porque la potencia de las señales de entrada es constante; sin embargo, en una situación real el umbral debe ser dinámico y auto-ajustarse a la potencia de éstas.

El filtro del bucle que limita el ancho de banda de correlación se ha diseñado mediante un filtro IIR

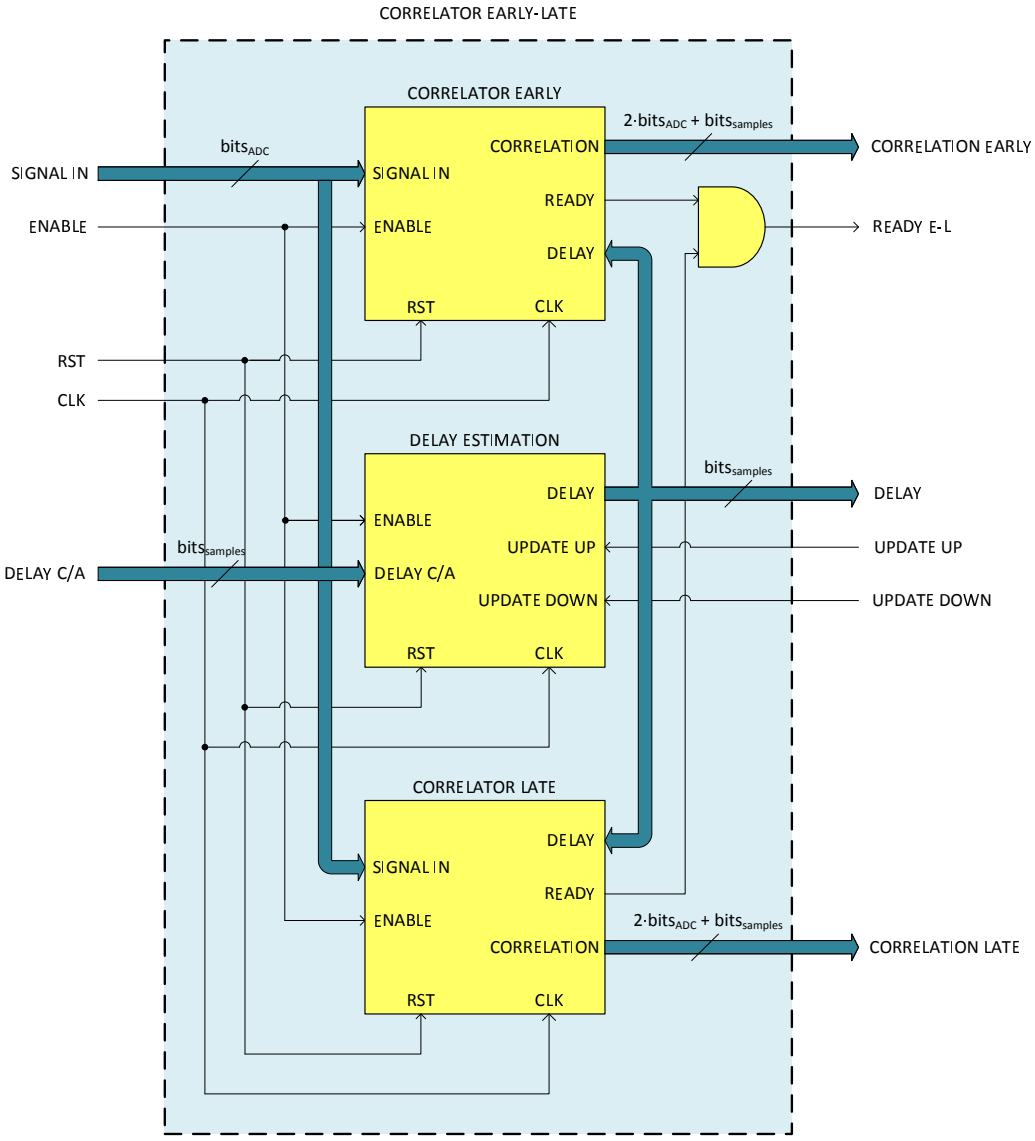


Figura 3.11: Diagrama de bloques VHDL del correlador *early-late*.

(Infinite Impulse Response) paso-bajo. Para que el filtro siga correctamente las variaciones de la función del discriminador la frecuencia de corte del filtro se ha establecido en $f_c = 100$ KHz. El filtro diseñado presenta la función de transferencia de la ecuación (3.3). La implementación de filtro se ha realizado en el módulo Loop Filter, y sus coeficientes han sido cuantificados en formato Q = [16, 0].

$$H(z) = \frac{Y(z)}{X(z)} = \frac{3.132 \cdot 10^{-5} + 3.132 \cdot 10^{-5} z^{-1}}{1 + 0.9937 z^{-1}} \tag{3.3}$$

Para comprobar que la etapa de tracking está funcionando adecuadamente se ha realizado una simulación funcional en la que la señal de entrada presenta una fase inicial nula y un retardo dinámico. El retardo inicial de la señal de entrada es $\tau_{LOS} = 131$ muestras; cada 20 épocas de señal el retardo sufre una variación de una muestra del siguiente modo: inicialmente se ha incrementado el retardo hasta $\tau_{LOS} = 135$ muestras, a continuación se ha disminuido hasta $\tau_{LOS} = 127$ muestras, y finalmente se ha incrementado de nuevo hasta $\tau_{LOS} = 131$ muestras. Los resultados de la simulación pueden observarse en la Figura 3.13. Cuando concluye la etapa de sincronización C/A (activación a nivel alto de la señal Ready C/A) la estimación gruesa del retardo proporcionada ($\hat{\tau}_0$) actúa como punto de inicio de la estimación continua

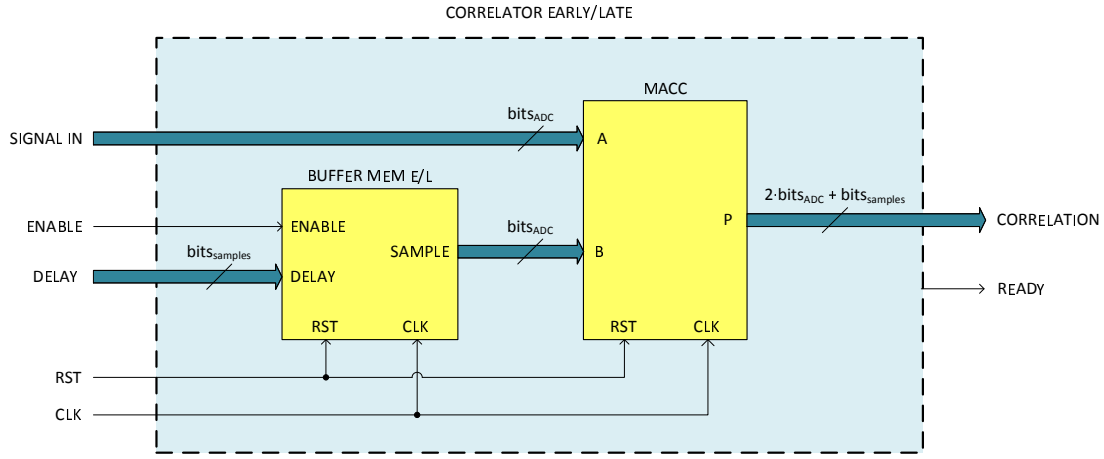


Figura 3.12: Diagrama de bloques VHDL del correlador *E/L*.

del retardo. Cada vez que el retardo de la señal de entrada experimenta una variación se produce un pico en la señal Difference EL que es filtrado mediante el componente Loop Filter dando lugar a la señal Discriminator. Cuando la señal Discriminator supera el umbral Threshold EL previamente establecido se genera un pulso en las señales Update Up o Update Down según el signo del discriminador y la estimación del retardo es actualizada. Los pulsos a nivel alto de la señal Update indican que se está actualizando la estimación del retardo. Las actualizaciones del retardo duran dos épocas de señal: tras la primera época se genera un pico en la señal Difference EL detectándose que ha habido una variación del retardo, pero debido al filtro, el discriminador no supera el umbral hasta el final de la segunda época.

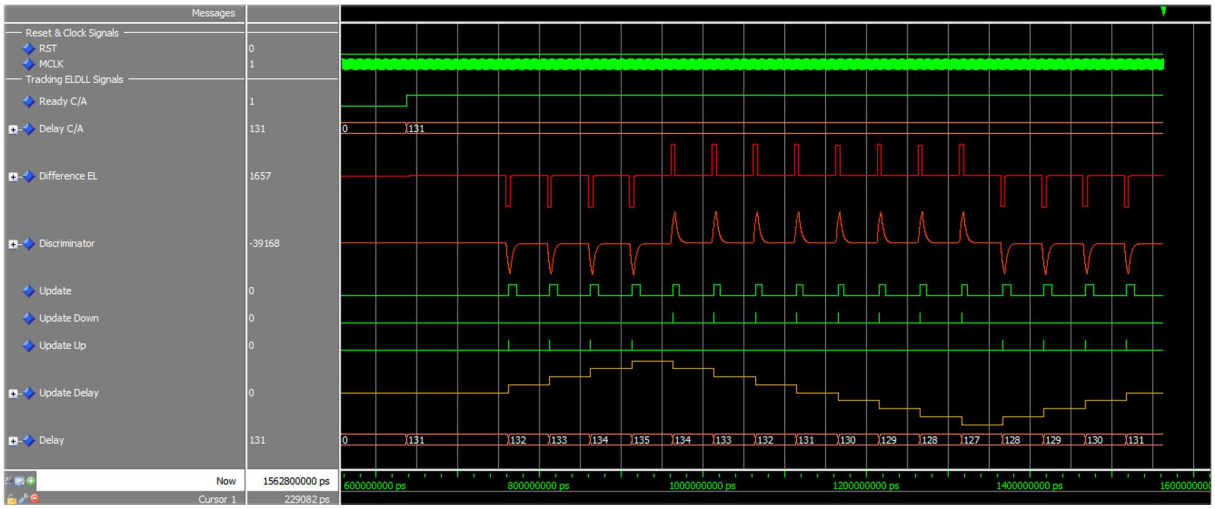


Figura 3.13: Cronograma funcional de la etapa de tracking ELDLL.

3.2.3 Demodulación DSSS

El IP Demodulation DSSS realiza la demodulación DSSS (Signal DSSS) de la señal de entrada (Signal In) a partir de la estimación continua del retardo (Delay) generada por el IP Tracking ELDLL a partir del momento en que éste está disponible (Enable). El IP Demodulation DSSS está formado por el módulo Buffer Mem PRN y el core Multiplier. La Figura 3.14 muestra el diagrama de bloque del IP Demodulation DSSS. Cuando la señal de entrada comienza a demodularse, la señal Ready DSSS se activa para iniciar la estimación de fase.

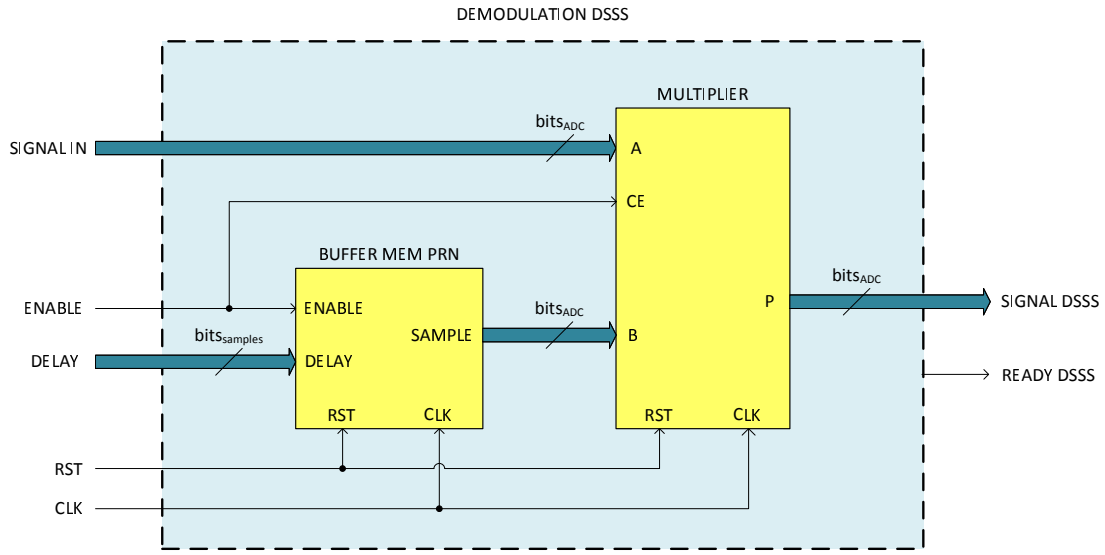


Figura 3.14: Diagrama de bloques VHDL de la etapa de demodulación DSSS.

El módulo Buffer Mem PRN genera una réplica de la secuencia PRN esperada en fase con la misma para demodular la señal de entrada. El Buffer Mem PRN está formado por el core Memory PRN (basado en la primitiva *RAMB16BWER*), el cual implementa la memoria local que contiene la secuencia PRN empleada en el ensanchamiento de la señal emitida. La secuencia PRN es generada en fase a partir de la estimación continual del retardo. El core Multiplier desensancha la señal de entrada multiplicando las muestras de la señal de entrada con las muestras de la secuencia PRN para devolver a la señal de entrada a su espectro original. Al igual que en el IP Tracking ELDLL, la señal de entrada debe retardarse cuatro ciclos para sincronizarla con la secuencia PRN, ya que ésta también procede de una memoria con dos ciclos de latencia a los que se suman los dos ciclos de latencia introducidos por el IP Synchronization C/A. Cuando la señal de entrada comienza a ser demodulada la señal Ready DSSS se activa a nivel alto para habilitar la estimación de fase.

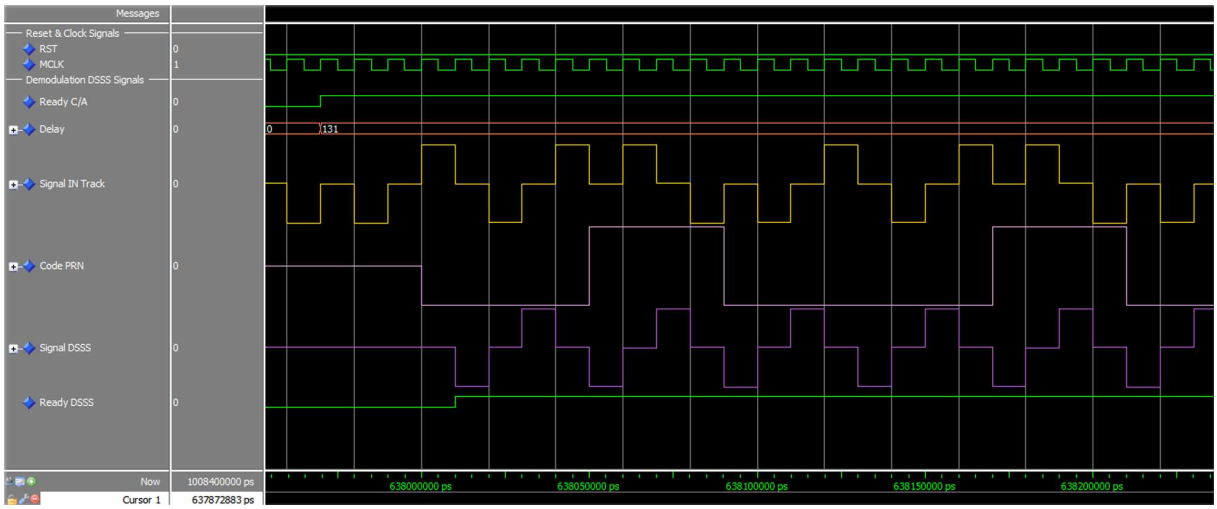


Figura 3.15: Cronograma funcional de la etapa de demodulación DSSS.

En la Figura 3.15 se muestra un fragmento ampliado del cronograma funcional obtenido al realizar una simulación en la que la señal de entrada tiene un retardo $\tau_{LOS} = 131$ muestras. Como se puede ver, una vez finalizada la etapa de sincronización C/A, se emplea el retardo estimado por el IP Tracking ELDLL para generar una réplica de la secuencia PRN esperada en fase con la señal de entrada. Puesto

que el retardo estimado coincide con el retardo de la señal de entrada, ésta es demodulada correcta y coherentemente, obteniéndose una señal senoidal de cuatro muestras por ciclo.

3.2.4 Estimación de fase

El IP Phase Estimation se encarga de proporcionar la estimación de fase de la señal de entrada (Signal In) una vez que ésta está siendo demodulada (Enable). Dicha estimación es realizada a partir de los módulos Demodulation I/Q, Filter I/Q, y Phase Extractor. En la Figura 3.16 puede verse la arquitectura interna del IP Phase Estimation.

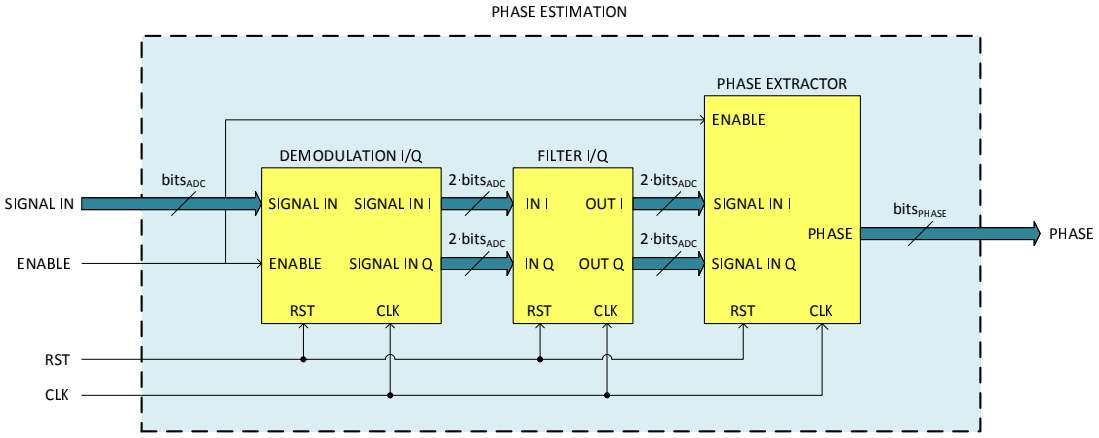


Figura 3.16: Diagrama de bloques VHDL del estimador de fase.

El módulo Demodulation I/Q está formado por el componente Buffer Mem I/Q, y por los cores Multiplier I y Multiplier Q. El componente Buffer Mem I/Q está formado a su vez por los cores Memory I y Memory Q (basados en la primitiva *RAMB16BWER*) que implementan las memorias locales que contienen las señales de referencia I/Q. Los cores Multiplier I y Multiplier Q realizan la demodulación I/Q de la señal de entrada (Signal In) multiplicando a ésta por las señales de referencia I/Q generadas por el componente Buffer Mem I/Q, y dando como resultado las señales Signal In I y Signal In Q. La Figura 3.17 se representa la estructura del módulo Demodulation I/Q.

Como ya se comentó en el apartado 2.3.2.4, las frecuencias de la señal de entrada y de las señales de referencia I/Q al ser generadas por osciladores distintos no son idénticas. Analizando diferentes osciladores comerciales actuales, es típico encontrar osciladores con una estabilidad en frecuencia $\varepsilon_{freq} = \pm$ 2 ppm, lo que dada la frecuencia de modulación empleada equivale $\pm$ 50 Hz. Los peores casos se dan cuando el oscilador de emisión y el oscilador I/Q presentan el valor máximo de ε_{freq} y de signo contrario, casos en los cuales $\Delta f = \pm\, 2 \cdot \varepsilon_{freq} = \pm$ 4 ppm $= \pm$ 100 Hz. La diferencia de frecuencias entre ambos osciladores $\Delta f = 100$ Hz se ha introducido en las señales de referencia I/Q.

Las componentes de alta frecuencia de las señales Signal In I y Signal In Q son eliminadas por un filtro IIR paso-bajo con una frecuencia de corte $f_c = 100$ KHz y que presenta la función de transferencia de la ecuación (3.4). Aunque a priori la frecuencia de corte del filtro no es muy restrictiva, ya que hay que separar una componente de baja frecuencia (Δf) de una componente de alta frecuencia ($2 \cdot f_m + \Delta f$), el filtro debe ser lo suficientemente rápido como para seguir las variaciones del retardo de la señal de entrada. En el módulo Filter I/Q se han implementado dos filtros, uno para cada señal, y sus coeficientes han sido cuantificados en formato Q = [16, 0].

$$H(z) = \frac{Y(z)}{X(z)} = \frac{3.132 \cdot 10^{-3} \cdot (1 + z^{-1})}{1 + 0.9937 z^{-1}} \tag{3.4}$$

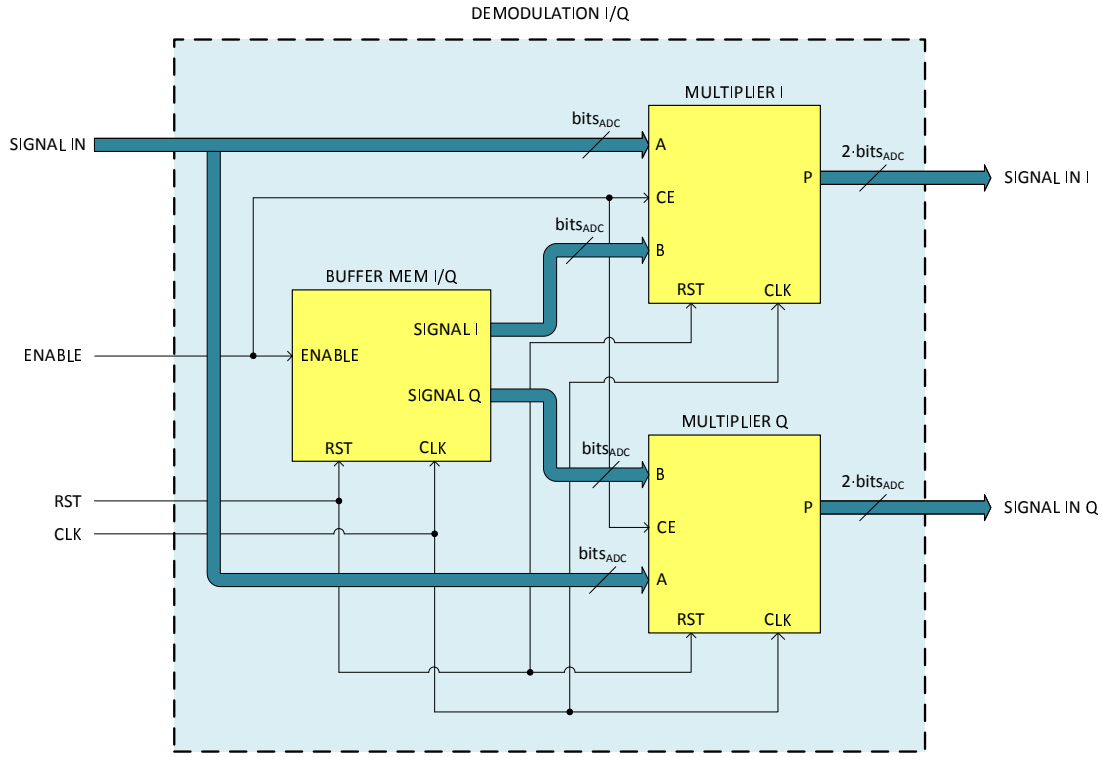


Figura 3.17: Diagrama de bloques VHDL del demodulador I/Q.

El módulo Phase Extractor consiste en el core Arctan, el cual está basado en el IP CORDIC que implementa el algoritmo generalizado que le da su nombre *(COordinate Rotational DIgital Computer)*. De las numerosas funciones trigonométricas que este IP es capaz de implementar, se ha configurado como arcotangente para poder obtener la estimación de fase a partir de las señales Signal In I y Signal In Q una vez han sido filtradas. La estimación de fase es proporciona en radianes dentro del intervalo $\pm\,\pi$ rad en coma fija, con formato $Q_{PHASE} = [16,\,3]$, y en complemento a 2.

Una vez la señal de entrada ($\varphi_0 = 0^{\text{o}}$ y $\tau_{LOS} = 131$ muestras) ha sido demodulada coherentemente mediante DSSS, se ha realizado una simulación para comprobar que la fase estimada es la que presenta la señal de entrada. Dado que un ciclo de la señal de entrada consta de 4 muestras, cada muestra de retardo introduce un desfase de $\pi/2$ rad sobre la fase inicial φ_0 de la señal emitida. El retardo de la señal de entrada $\tau_{LOS} = 131 \equiv 32 \cdot 4 + 3$ muestras equivale a 32 ciclos completos y 3 muestras de retardo, por lo que la fase de la señal de entrada cuando alcanza el receptor es $\varphi_{LOS} = \varphi_0 + 3 \cdot 90^{\text{o}} = 270^{\text{o}}$. En la Figura 3.18 se representa el cronograma funcional obtenido tras realizar la simulación. En primer lugar aparece la señal de entrada demodulada por sus componentes I/Q, y posteriormente éstas ya filtradas. En segundo lugar, sobre las señales filtradas se lleva a cabo la estimación de fase. Debe tenerse en cuenta a la hora de interpretar los resultados que la fase estimada se encuentra en formato Q_{PHASE}, por lo que para obtener el valor real de la fase ésta debe ser "descuantificada" con dicho formato. Esta operación es llevada a cabo off-line en MATLAB, y la fase medida en radianes ha sido convertida a grados obteniéndose una estimación de fase con un error medio absoluto $|\overline{\varepsilon}_{\hat{\varphi}}| = 0.794^{\text{o}}$, con una desviación típica $\sigma_{\hat{\varphi}} = 0.109^{\text{o}}$.

En este punto del diseño se ha caracterizado el error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la estimación de fase para cada una de las posibles fases estimables de las señales de entrada: 0º, 90º, 180º, y 270º. Sólo se pueden estimar estas fases debido a que la resolución de las señales de entrada y de las memorias locales es de 4 muestras/chip y la fase inicial es nula. En las Figuras 3.19(a) y 3.19(b) se muestran los resultados obtenidos de las medidas realizadas. Se puede ver como para $\varphi_{LOS} = 0^{\text{o}}$, 270^{o} el error medio se dispara hasta $\sim 0.8^{\text{o}}$, mientras que para $\varphi_{LOS} = 90^{\text{o}}$, 180^{o} el error medio es de apenas

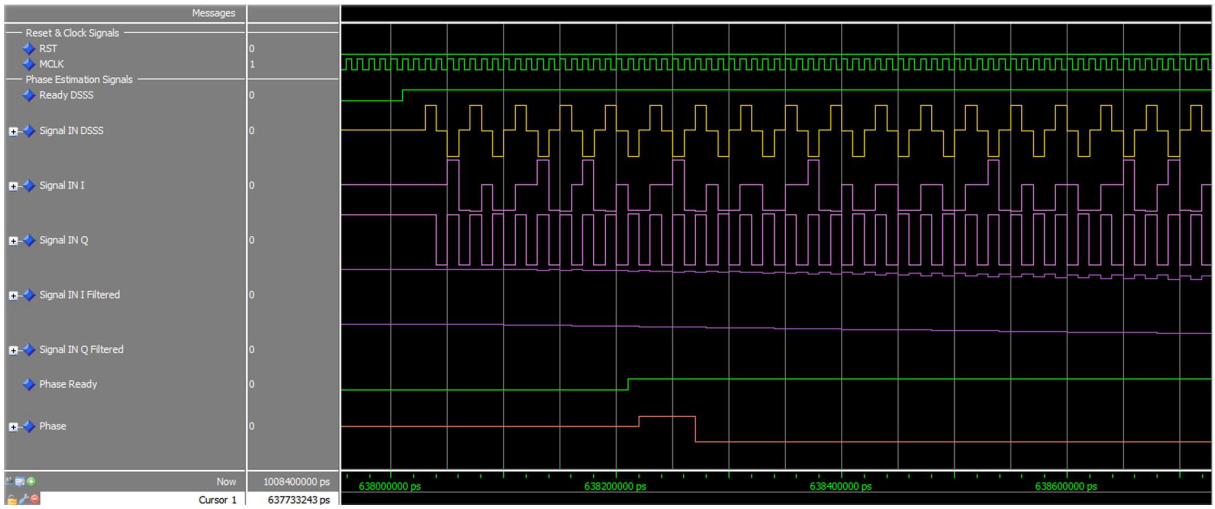
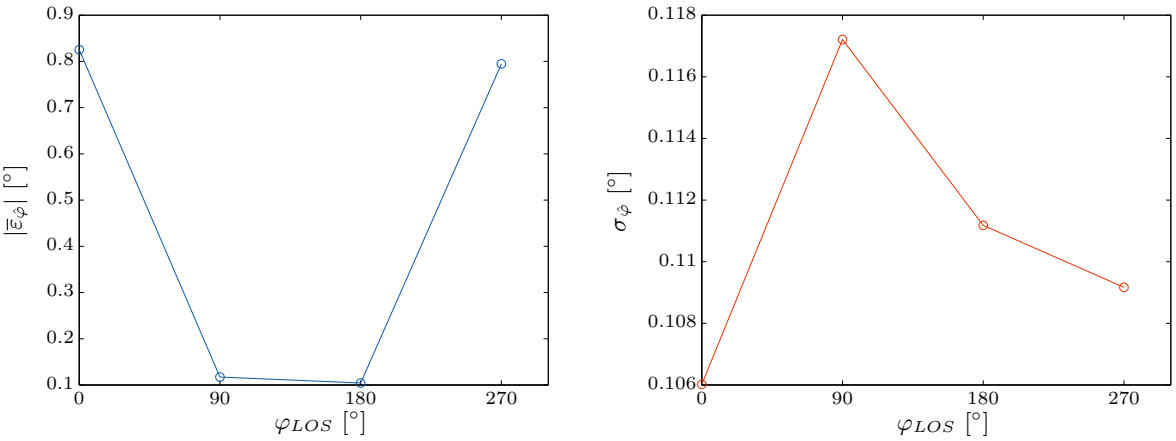


Figura 3.18: Cronograma funcional de la etapa de estimación de fase.

0.1º. En cuanto a la desviación típica, los valores obtenidos para las distintas fases presentan una baja variabilidad, siendo bastante similares entre sí.



(a) Error medio.

(b) Desviación típica.

Figura 3.19: Error medio y desviación típica de la estimación de fase individual en condiciones ideales.

3.2.5 Estimación diferencial de fase

Tras verificar que los IP's que conforman el módulo PMS funcionan correctamente, y que éste estima la fase de la señal de entrada con una adecuada precisión, se ha llevado a cabo una simulación para comprobar que el módulo Differential Phase funciona bien. Las señales de entrada Signal I y Signal J presentan una fase inicial $\varphi_0 = 0^o$ y un retardo aleatorio: $\tau_{LOS_i} = 131$ muestras y $\tau_{LOS_j} = 33$ muestras respectivamente. Como ya se comentó al final del apartado anterior, el retardo de las señales modifica la fase inicial de éstas al alcanzar los receptores cuando el retardo no es múltiplo de cuatro. Teniendo en cuenta que los retardos son $\tau_{LOS_i} = 131 \equiv 32 \cdot 4 + 3$ muestras y $\tau_{LOS_j} = 33 \equiv 8 \cdot 4 + 1$ muestras, las fases realmente estimadas deben ser $\varphi_{LOS_i} = \varphi_0 + 270^o = 270$ y $\varphi_{LOS_j} = \varphi_0 + 90^o = 90^o$. De este modo, la diferencia de fase esperada es $\varphi_{LOS_{ij}} = 180^o$.

La Figura 3.20 representa el cronograma funcional obtenido en la simulación. Ambas señales presentan el retardo previsto $\hat{\tau}_i = 131$ muestras y $\hat{\tau}_j = 33$ muestras. Debido a que las fases estimadas no son

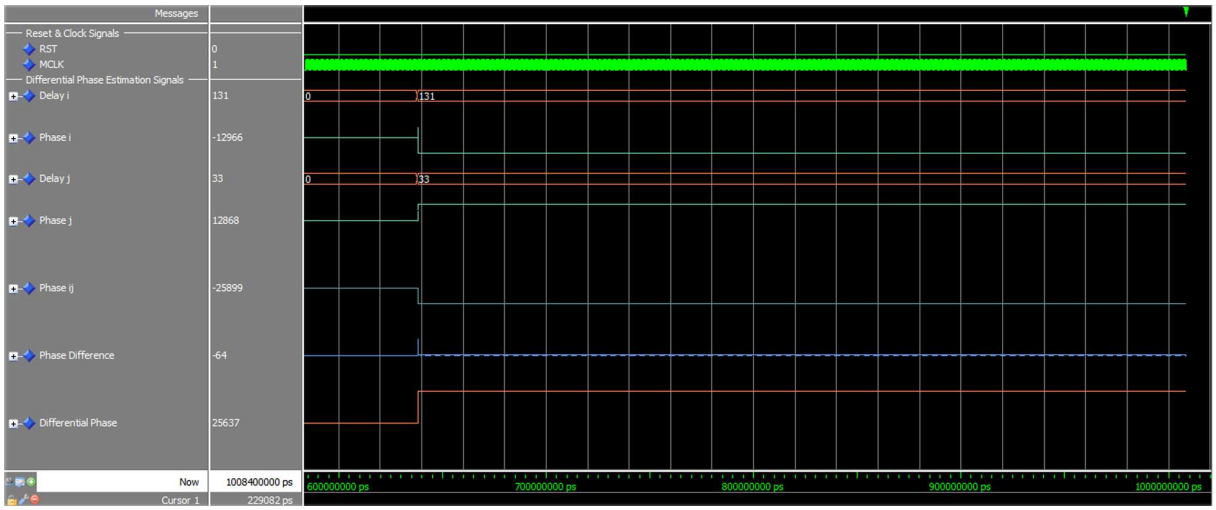


Figura 3.20: Cronograma funcional de la estimación diferencial de fase.

exactamente -$\pi/2$ y $\pi/2$, la fase diferencial es ligeramente inferior a -π, y tal como se puede ver, es corregida por el módulo Differential Phase obteniéndose una fase diferencial próxima a π. La diferencia de fase estimada presenta un error medio $|\overline{\varepsilon}_{\hat{\varphi}_{ij}}| = 0.904$ º, y $\sigma_{\hat{\varphi}_{ij}} = 0.225$ º.

3.2.6 Estimación diferencial de distancia

Finalmente, para verificar que el diseño completo DDMS funciona adecuadamente, se ha realizado una simulación en las mismas condiciones que en el apartado anterior ($\varphi_0 = 0$º, $\tau_{LOS_i} = 131$ muestras, y $\tau_{LOS_j} = 33$ muestras), pero incorporando en este caso el módulo final Phase - Distance para obtener la estimación diferencial de distancia.

En la Figura 3.21 se muestra el cronograma funcional obtenido en la simulación. Ambas señales presentan el retardo previsto $\hat{\tau}_i = 131$ muestras y $\hat{\tau}_j = 33$ muestras. La distancia diferencial estimada presenta un error medio $|\overline{\varepsilon}_{\hat{d}_{ij}}| = 30.159$ mm, y $\sigma_{\hat{d}_{ij}} = 7.508$ mm. Los valores de error medio y desviación típica de fase medidos en el apartado anterior han sido convertidos a distancia obteniéndose el mismo resultado que en esta simulación, lo cual indica que en la cuantificación de la sensibilidad $K_{d/\varphi}$ no se ha perdido nada de precisión.

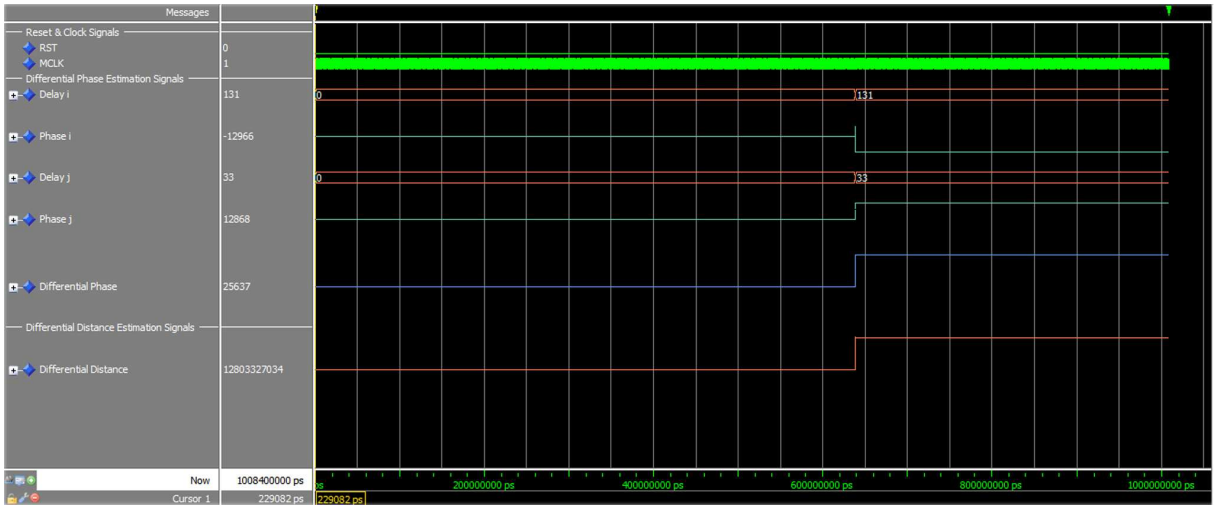


Figura 3.21: Cronograma funcional de la estimación diferencial de distancia.

Capítulo 4

Resultados y validación

Tras comprobar que la implementación de la arquitectura de medida de distancia diferencial funciona idealmente de forma satisfactoria, ésta ha sido sometida a numerosas pruebas experimentales para verificar y validar su correcta funcionalidad teniendo en cuenta las características no ideales de las señales de entrada que podrían darse en un entorno real (CBL). En dicho entorno las señales de entrada presentan un retardo dinámico ($\tau_{LOS}(t)$) debido al movimiento del robot móvil y a errores de frecuencia entre los osciladores del emisor y receptores (Δf), y además se encuentran afectadas por ruido en tensión e interferencias por multicamino. Para caracterizar el efecto de las fuentes de no idealidad se ha evaluado su impacto sobre la medida diferencial de distancia primero por separado y después conjuntamente, a partir de simulaciones temporales.

4.1 Condiciones ideales

En la sección 3.2 se vio que la implementación realizada del sistema de medida diferencial de distancia era correcta funcionalmente en condiciones ideales. Sin embargo, para confirmar que el diseño es completamente correcto debe garantizarse que una vez implementado sobre la FPGA cumple las restricciones temporales impuestas, esto es, trabajar con un periodo de reloj máximo $T_{CLK,máx} = 10.0$ ns.

La simulación temporal ha sido llevada a cabo bajo las mismas condiciones que la simulación funcional realizada en el apartado 3.2.6: una fase inicial $\varphi_0 = 0^o$, y un retardo $\tau_{LOS_i} = 131$ muestras y $\tau_{LOS_j} = 33$ muestras. En la Figura 4.1 se representa el cronograma temporal con los resultados obtenidos. Los valores de error medio y desviación típica obtenidos son ligeramente superiores en comparación a los obtenidos en la simulación funcional: $|\overline{\varepsilon}_{\hat{d}_{ij}}| = 37.944$ mm, y $\sigma_{\hat{d}_{ij}} = 7.889$ mm, con una variación entre sí inferior a 10 mm ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y 1 mm ($\sigma_{\hat{d}_{ij}}$).

Posteriormente se ha medido el error medio y la desviación típica sobre todas las posibles diferencias de fase estimables: 0^o, 90^o, 180^o, y 270^o, a partir de todas las posibles combinaciones de retardo entre las señales de entrada. En las Figuras 4.2(a) y 4.2(b) se muestra el error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia obtenidos para las cuatro posibles diferencias de fase. Puede observarse que el error medio para una fase diferencial de 180^o ($|\overline{\varepsilon}_{\hat{d}_{ij}}| \approx 38$ mm) es muy superior respecto al error medio del resto de diferencias de fase. Ello es debido a que una de las fases individuales (φ_{LOS_i} o φ_{LOS_j}) para estimar una fase diferencial $\varphi_{LOS_{ij}} = 180^o$ (270^o - 90^o, 180^o - 0^o) presenta siempre un elevado error medio en la estimación tal como se vio en la Figura 3.19(a) del apartado 3.2.4 ($\varphi_{LOS} = 0^o$, 270^o). En cambio, cuando $\varphi_{LOS_{ij}} = 0^o$ el error medio es nulo, ya que la estimación diferencial es generada por dos estimaciones de fase iguales, y por tanto, el error se anula. Las estimaciones de las fases

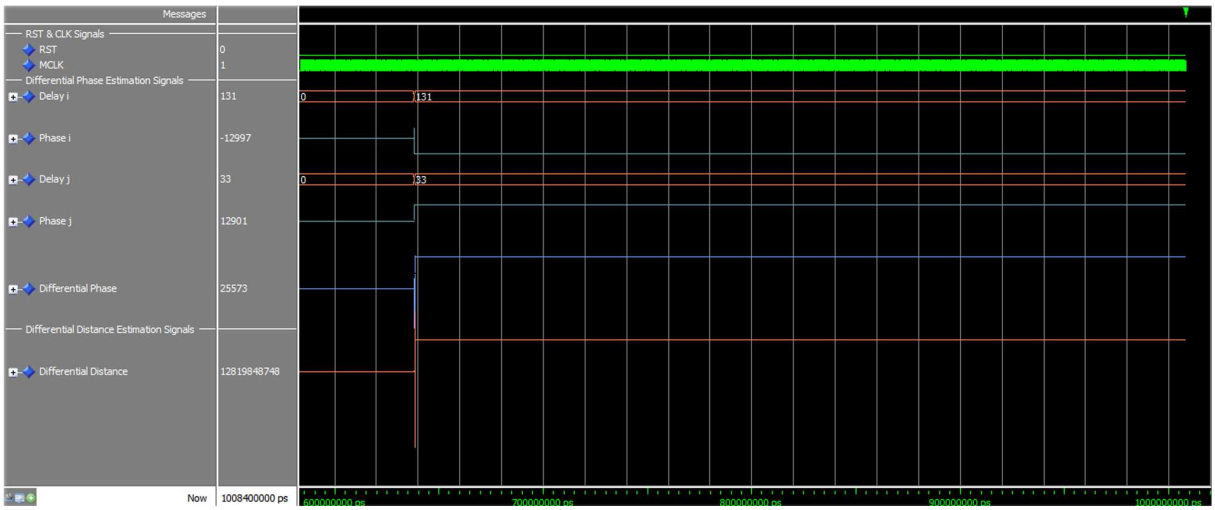
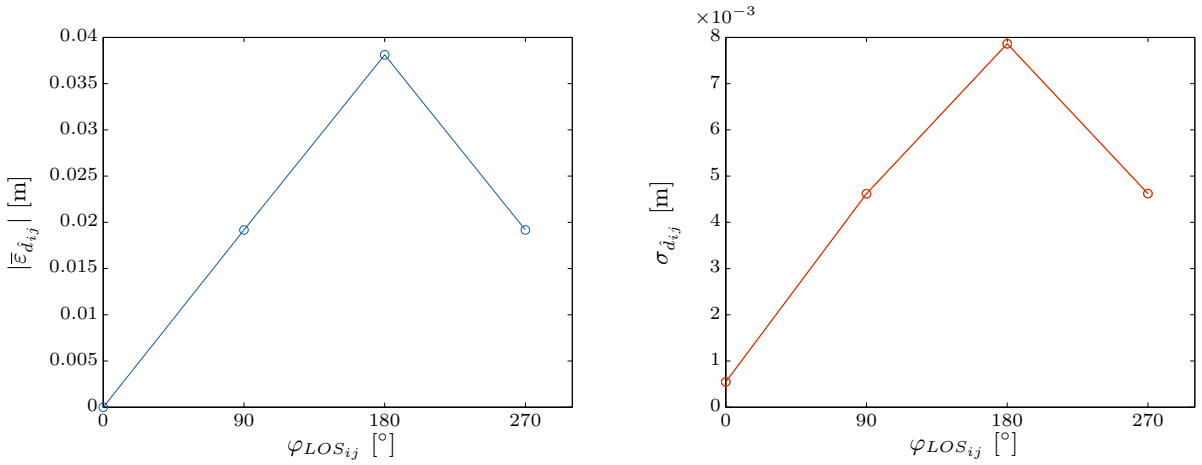


Figura 4.1: Cronograma temporal de la estimación diferencial de distancia en condiciones ideales.

diferenciales $\varphi_{LOS_{ij}} = 90^{o}$, 270^{o} tienen un error medio intermedio ($|\overline{\varepsilon}_{\hat{d}_{ij}}| \approx 19$ mm) entre el error de $\varphi_{LOS_{ij}} = 0^{o}$ y $\varphi_{LOS_{ij}} = 180^{o}$, pues las fases individuales con un error medio más crítico sólo intervienen en la mitad de casos posibles para estimarlas. Por otra parte, la desviación típica presenta una tendencia muy similar a la del error medio, pues para $\varphi_{LOS_{ij}} = 90^{o}$, 270^{o} ésta tiene un valor intermedio al medido para $\varphi_{LOS_{ij}} = 0^{o}$ y $\varphi_{LOS_{ij}} = 180^{o}$, aunque la variabilidad entre los distintos valores es muchísimo menor que en el caso del error medio.



(a) Error medio.

(b) Desviación típica.

Figura 4.2: Error medio y desviación típica de la estimación diferencial de distancia en condiciones ideales.

4.2 Retardo dinámico en las señales de entrada

El desplazamiento del robot móvil dentro la CBL hace que la distancia entre éste y una pareja de receptores varíe permanentemente. La variación de distancia y los errores en frecuencia entre emisor y receptores a su vez provocan que el retardo de la componente directa de la señal que alcanza el receptor también varíe. La coherencia en la etapa de demodulación DSSS se mantiene prácticamente constante ya que el retardo estimado es actualizado continuamente por la etapa de tracking ELDLL. Se han realizado dos tipos de pruebas con el objetivo de ver cómo afecta la dinámica del robot móvil a la estimación diferencial de distancia. En la primera prueba se ha modificado únicamente el retardo de una de las señales de entrada; en la segunda se ha aplicado a ambas señales la misma variación en el retardo.

4.2.1 Retardo dinámico en una única señal de entrada

La Figura 4.3 representa el cronograma temporal obtenido para dos señales con fase inicial $\varphi_0 = 0^o$, y un retardo $\tau_{LOS_i} = 131 \rightarrow 135$ muestras y $\tau_{LOS_j} = 33$ muestras respectivamente; cada 50 épocas de señal se incrementa en una muestra τ_{LOS_i}, por lo que, dada la ecuación (4.1), el retardo varía a una velocidad $\Delta\tau = 79.365\ \mu s/\text{s}$. Cuando se produce una variación en el retardo de la señal de entrada se genera un transitorio en la estimación diferencial de distancia durante las dos épocas de señal que éste tarda en actualizarse. Sobre cada uno de los tramos de la estimación diferencial de distancia asociados a las diferentes fases diferenciales se ha medido el error medio absoluto y la desviación típica una vez se ha estabilizado la respuesta del filtro. En las Figuras 4.4(a) y 4.4(b) se muestran ambos resultados medidos para las cuatro posibles diferencias de fase. Se observa que el error medio ha sufrido un incremento mínimo ($\sim$ 5 mm) respecto a los resultados obtenidos en condiciones ideales, mientras que la desviación típica, pese a mantener la misma tendencia, ha reducido considerablemente su dispersión. El retardo dinámico en una única señal de entrada apenas tiene efecto sobre la estimación diferencial de distancia salvo en la dispersión de la desviación típica, que se reduce en más de 7 mm.

$$\Delta\tau = \frac{\partial\tau}{\partial t} = \frac{1}{n_{épocas}} \cdot \frac{1}{t_{signal}} \cdot T_s \quad [s/s] \tag{4.1}$$

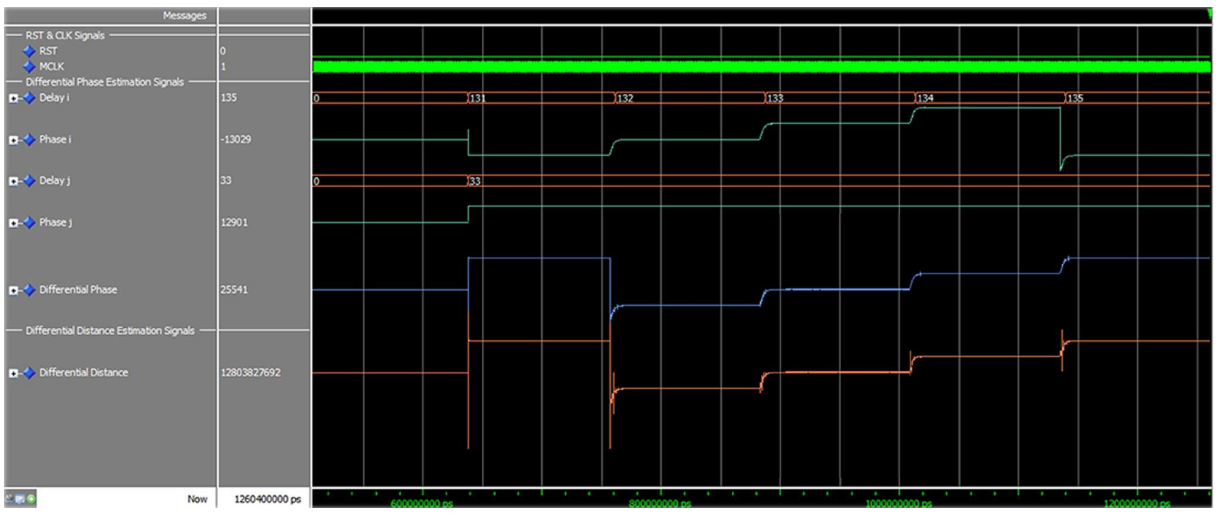
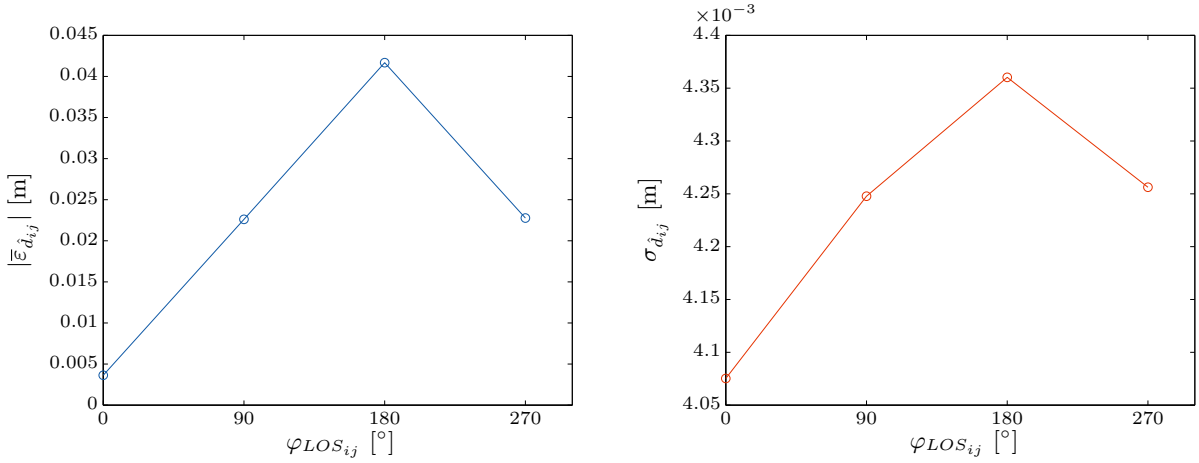


Figura 4.3: Cronograma temporal de la estimación diferencial de distancia con retardo dinámico en una única señal de entrada.



(a) Error medio.

(b) Desviación típica.

Figura 4.4: Error medio y desviación típica de la estimación diferencial de distancia con retardo dinámico en una única señal de entrada.

4.2.2 Retardo dinámico en ambas señales de entrada

En la segunda prueba realizada se ha variado muestra a muestra cada 20 épocas de señal el retardo de ambas señales de entrada del mismo modo: partiendo de un retardo inicial $\tau_{LOS_i} = 131$ muestras y $\tau_{LOS_j} = 33$ muestras, se ha incrementando hasta alcanzar un retardo $\tau_{LOS_i} = 135$ muestras y $\tau_{LOS_j} = 37$ muestras, a continuación se ha decrementado hasta un retardo $\tau_{LOS_i} = 127$ muestras y $\tau_{LOS_j} = 29$ muestras, y finalmente se ha vuelto a aumentar el retardo hasta recuperar el inicial, variando a una velocidad $\Delta\tau = 198.412\ \mu s/\text{s}$ (4.1). De este modo se emularía el movimiento del robot móvil desplazándose equidistante entre ambos receptores. En la Figura 4.5 se representa el cronograma temporal obtenido tras la simulación. Se observa que los transitorios que se generan en la estimación de fase con cada variación del retardo aparecen levemente en la estimación diferencial de distancia ya que al ser simultáneos en ambas señales se cancelan. Por último puede verse como el componente Phase Correction del módulo Differential Phase corrige la estimación de fase diferencial eliminando la variación entre $\pm\ \pi$ rad. La medida diferencial de distancia presenta un error medio $|\overline{\varepsilon}_{\hat{d}_{ij}}| = 5.427$ mm, y una desviación típica $\sigma_{\hat{d}_{ij}} = 44.626$ mm.

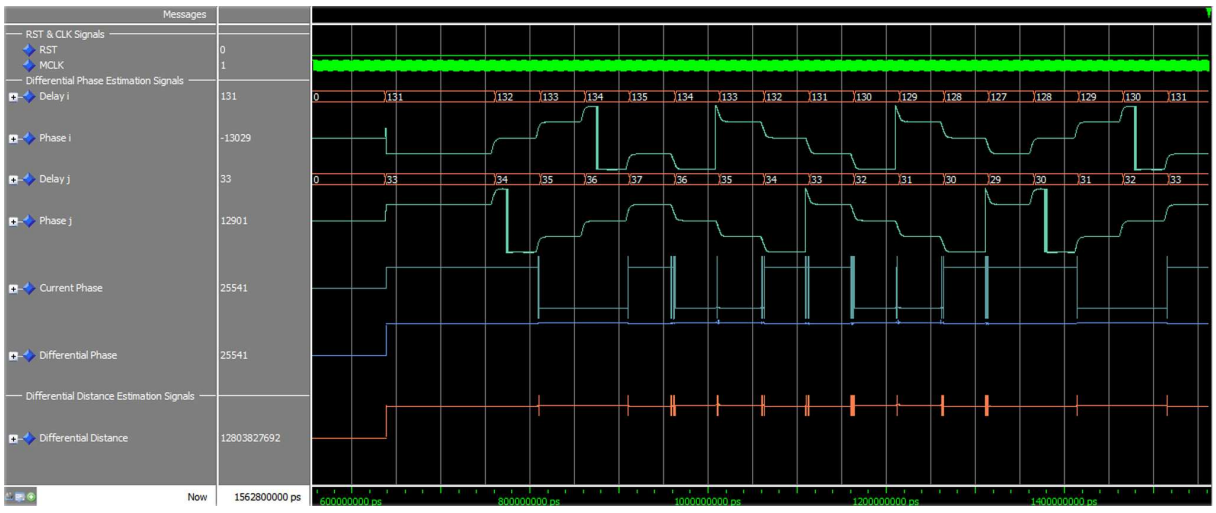
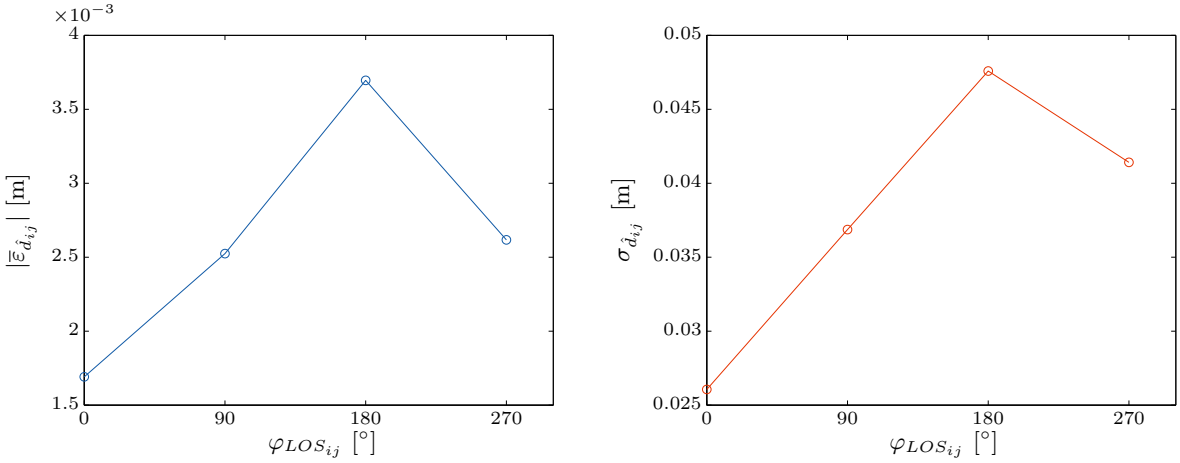


Figura 4.5: Cronograma temporal de la estimación diferencial de distancia con retardo dinámico en ambas señales de entrada.



(a) Error medio.

(b) Desviación típica.

Figura 4.6: Error medio y desviación típica de la estimación diferencial de distancia con retardo dinámico en ambas señales de entrada.

Una vez medidos los parámetros característicos para una determinada diferencia de fase se ha evaluado el error medio y la desviación típica sobre todas las posibles fases diferenciales estimables. En las Figuras 4.6(a) y 4.6(b) se muestra el error medio absoluto ($|\bar{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia obtenidos para las cuatro posibles diferencias de fase. Puede verse como el error medio se ha reducido en casi un 90 % mientras que la desviación típica ha aumentado en más de 5 veces en comparación a los resultados obtenidos en condiciones ideales o con variación de retardo en una única señal. Tal como se aprecia en la Figura 4.5 la estimación de fase diferencial inicial (Phase ij) puede variar entre $\varphi_{LOS_{ij}}$ y $\varphi_{LOS_{ij}}$ - 2π en función de la fase individual de cada señal, por lo que ésta debe ser corregida por el componente Phase Correction del módulo Differential Phase. Ello provoca que la estimación diferencial de distancia final no sea siempre la misma, sino unas veces superior, y otras inferior. De este modo el error medio se reduce, pero al mismo tiempo la desviación típica se incrementa.

4.3 Ruido en las señales de entrada

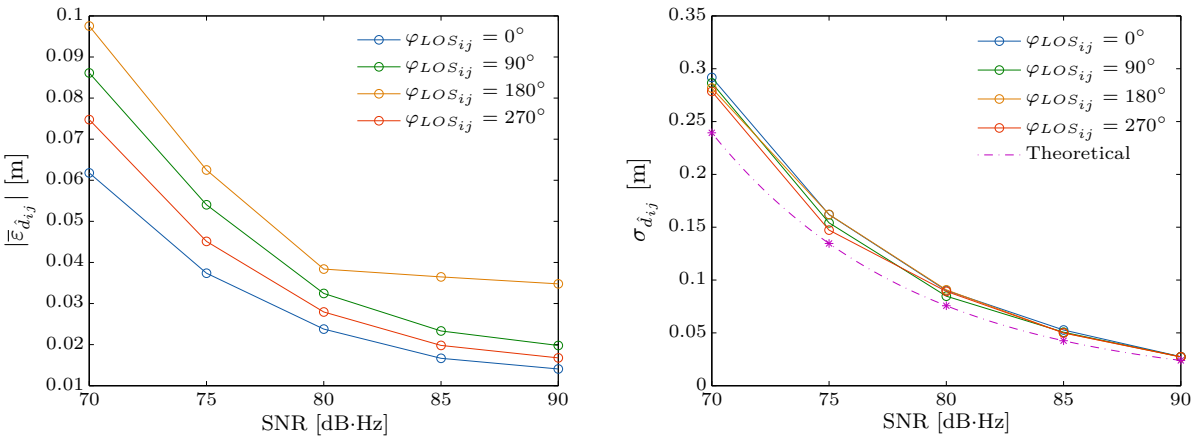
La principal fuente de ruido del LPS reside en la electrónica de acondicionamiento de las señales que alcanzan los receptores. Dadas las dimensiones de la CBL, y teniendo en cuenta las características del enlace óptico IR, el SNR de las señales a la salida de los receptores (definido por la ecuación (4.2)) varía dentro del rango [70 - 90] dB·Hz. Para evaluar el efecto del SNR sobre la estimación diferencial de distancia, a las señales de entrada al sistema de medida, hasta el momento ideales, se les ha añadido AWGN *(Additive White Gaussian Noise)*. Este ruido está definido por su densidad de potencia $N_o/2$, la cual es calculada en función del SNR y la potencia de señal P_s a partir de la ecuación (4.2), y de donde se deriva además su varianza σ_n^2 según la ecuación (4.3). La varianza está definida sobre $f_s/2$ debido a que es la frecuencia a la que se encuentra limitado el ruido por el filtro anti-aliasing previo a la digitalización. Se han llevado a cabo dos tipos de pruebas: en la primera de ellas se ha introducido ruido únicamente en una de las señales de entrada al sistema manteniéndose la otra ideal; en la segunda prueba se ha añadido ruido sobre ambas señales de entrada considerándose distintas posiciones del robot móvil entre dos receptores a fin de emular el movimiento de éste por el interior de la CBL.

$$SNR = \lg\left(\frac{P_s}{N_o/2}\right) \quad [dB \cdot Hz] \tag{4.2}$$

$$\sigma_n^2 = N_o \cdot \frac{f_s}{2} \quad [V] \tag{4.3}$$

4.3.1 Ruido en un único receptor

Esta prueba pretende caracterizar la contribución del SNR de un único receptor sobre el error medio y desviación típica de la estimación diferencial de distancia. Las Figuras 4.7(a) y 4.7(b) representan el error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia obtenidos para las cuatro posibles diferencias de fase en función de distintos niveles de SNR en un único receptor.



(a) Error medio.

(b) Desviación típica.

Figura 4.7: Error medio y desviación típica de la estimación diferencial de distancia con ruido en un único receptor.

Puede verse como la mejora del SNR lleva asociada una mejora del error medio y de la desviación típica para las 4 posibles fases diferenciales. En las peores condiciones de SNR el error medio alcanza un valor máximo próximo a 0.1 m ($\varphi_{LOS_{ij}} = 180°$) con una variación de $\sim$ 10 mm entre las distintas fases

diferenciales. En cambio, en las mejores condiciones de SNR se logran errores inferiores a 15 mm. Sin embargo, para $\varphi_{LOS_{ij}} = 180°$, la mejora del error medio se detiene en torno a 38 mm a partir de SNR = 80 dB·Hz, pues es el error mínimo alcanzable para dicha diferencia de fases dadas las características del estimador por arcotangente tal como ya se vio en la sección 4.1.

Respecto a la desviación típica, los valores obtenidos para las distintas diferencias de fase (líneas sólidas) tienen una variabilidad inferior a 10 mm para un mismo nivel de SNR. Con las condiciones de SNR más desfavorables se tiene una desviación típica de 290 mm aproximadamente, la cual se consigue mejorar hasta en 10 veces para las mejores condiciones de SNR. En la Figura 4.7(b) se muestra además la desviación típica teórica (línea discontinua) de la estimación diferencial de distancia (4.4) en función del NEB *(Noise Equivalent Bandwidth)* y del nivel de SNR en el receptor ruidoso. La frecuencia de corte más restrictiva del sistema viene impuesta por la frecuencia de corte de los filtros del bucle de la etapa de tracking ELDLL y de demodulación I/Q (f_c = 100 KHz); puesto que ambos son filtros paso-bajo de orden 1, la expresión del NEB para este tipo de filtros viene dada por la ecuación (4.5). Para un SNR = 70 dB·Hz la desviación típica obtenida es ligeramente superior a la teórica ($\sim$ 50 mm), pero a medida que las condiciones de SNR van mejorando, la diferencia entre ambas desviaciones se va reduciendo hasta que para un SNR = 90 dB·Hz ambas son prácticamente iguales. A la vista de estos resultados, cabe destacar la gran proximidad que existe entre el comportamiento del sistema y comportamiento teórico esperado en cuanto desviación típica de la estimación diferencial de distancia se refiere.

$$\sigma_{\hat{d_{ij}}} = \sqrt{NEB \cdot \left(\frac{1}{SNR}\right)} \cdot \frac{360}{2\pi} \cdot K_{d/\varphi} \quad [m] \tag{4.4}$$

$$NEB = f_c \cdot \frac{\pi}{2} \quad [Hz] \tag{4.5}$$

4.3.2 Ruido en ambos receptores

En una situación real, el movimiento del robot móvil por el interior de la CBL provoca que la distancia entre emisor y receptores varíe constantemente, y en consecuencia el SNR de las señales de entrada al sistema. Dados dos receptores cualesquiera de la CBL, cuando el robot móvil se encuentra exactamente debajo de un receptor la distancia a dicho receptor es mínima y el acoplamiento óptico y por tanto el SNR son máximos (90 dB·Hz), mientras que la distancia al otro receptor es máxima y el SNR es mínimo (70 dB·Hz). En la Figura 4.8 se muestran dos receptores de la CBL (R_i y R_j) y el robot móvil en las tres posiciones más representativas entre ambos receptores. De acuerdo a la Figura 4.8, la componente directa aparece en verde cuando el SNR es máximo, en amarillo cuando el SNR es medio, y en rojo cuando el SNR es mínimo.

(a) El robot móvil se encuentra a la mínima distancia de R_i y a la máxima de R_j: SNR_i = 90 dB·Hz y SNR_j = 70 dB·Hz.

(b) El robot móvil se encuentra equidistante de R_i y R_j: $SNR_i = SNR_j$ = 80 dB·Hz.

(c) El robot móvil se encuentra a la máxima distancia de R_i y a la mínima de R_j: SNR_i = 70 dB·Hz y SNR_j = 90 dB·Hz.

En las Figuras 4.9(a) y 4.9(b) se muestra la evolución del error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) (líneas sólidas) de la medida diferencial de distancia obtenidos para las cuatro posibles diferencias de fase en función de las distintas posiciones del robot móvil entre los receptores R_i y R_j.

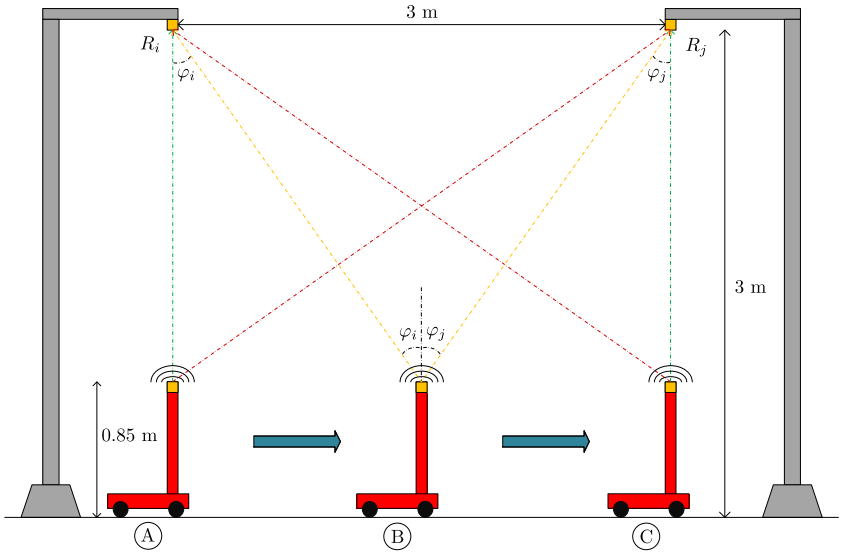
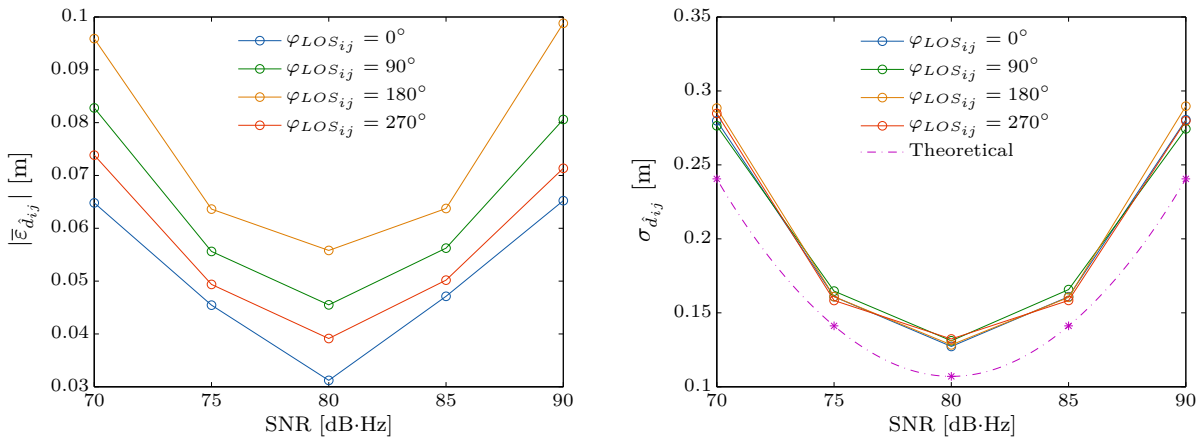


Figura 4.8: Esquema de posicionamiento del robot móvil entre dos receptores de la CBL.

Ambos resultados están en consonancia con los obtenidos en el apartado 4.3.1; presentan además una simetría respecto al punto medio entre ambos receptores (posición B, $SNR_{i,j} = 80$ dB·Hz), para el cual se obtienen los valores mínimos. También puede verse como cuando el robot móvil se aproxima hacia cualquiera de los receptores tanto el error medio como la desviación típica empeoran, pues la degradación del SNR respecto a un receptor tiene más influencia que la mejora de SNR respecto al otro, puesto que se trata de una relación logarítmica y no lineal.



(a) Error medio.

(b) Desviación típica.

Figura 4.9: Error medio y desviación típica de la estimación diferencial de distancia con ruido en ambos receptores.

En la Figura 4.9(b) aparece representada también la desviación típica teórica (línea discontinua) de la estimación diferencial de distancia (4.6) en función del NEB (4.5) y del nivel de SNR en cada receptor. La desviación típica obtenida para cualquiera de las cuatro fases diferenciales estimables se encuentra muy próxima a la teórica (< 50 mm), por lo que, tal como se comentó en el apartado 4.3.2, el comportamiento que muestra el sistema se acerca al teórico esperado.

$$\sigma_{\hat{d}_{ij}} = \sqrt{NEB \cdot \left(\frac{1}{SNR_i} + \frac{1}{SNR_j}\right)} \cdot \frac{360}{2\pi} \cdot \frac{d}{\varphi} \quad [m] \tag{4.6}$$

4.4 Presencia del multicamino en las señales de entrada

La principal propiedad del sistema de medida diferencial de distancia planteado en este trabajo es la mitigación de las interferencias por multicamino en la medida de lo posible. Con objeto de comprobar la eficacia de la implementación propuesta de la arquitectura frente a este tipo de interferencias, a las señales de entrada al sistema, constituidas hasta ahora exclusivamente por componentes LOS *(Line-Of-Sight)*, se le ha añadido a la señal que llega al receptor R_i una componente MP *(Multipath)* de manera que pueda verse con mayor claridad su efecto. En la componente MP añadida se ha variado tanto su potencia ($P_{MP}/P_{LOS} = [1 - 50]\,\%$) como su retardo ($\tau_{MP} = \tau_{LOS} + [1 - 4]$ muestras $\equiv$ [10 - 40] ns) respecto a la componente LOS.

La Figura 4.10 representa el cronograma temporal de la función de correlación perteneciente a la etapa de sincronización C/A para un multicamino con un retardo de una muestra (10 ns) y una potencia del 50 % respecto a la componente directa. Se puede ver claramente como la región del pico de correlación presenta una fuerte distorsión a causa de la proximidad e intensidad de la componente multicamino. Se trata de una situación de multicamino muy agresiva que se ha probado como caso más desfavorable. En el caso de multicaminos con una muestra de retardo y potencia superior al 50 % de la componente LOS pueden darse dos situaciones:

- El pico de correlación puede aparecer con una muestra de retraso, y por tanto, la estimación gruesa del retardo proporcionada por la etapa de sincronización C/A será incorrecta. Este problema es resuelto por el propio sistema, pues como se explicó en el apartado 3.2.1, mientras la estimación gruesa del retardo sea proporcionada con un error inferior a ± 1.0 chip, ésta puede ser corregida a continuación por la etapa de tracking ELDLL.
- Incluso cuando la estimación gruesa del retardo sea la correcta puede suceder que la función del discriminador de la etapa de tracking ELDLL supere el umbral de actualización de la estimación del retardo. En tal caso el retardo actualizado sería falso, lo que conllevaría una demodulación DSSS incoherente, y finalmente una estimación de fase y de distancia errónea. Este caso corresponde a un multicamino severo que, aunque no es típico de espacios interiores, no se puede resolver.

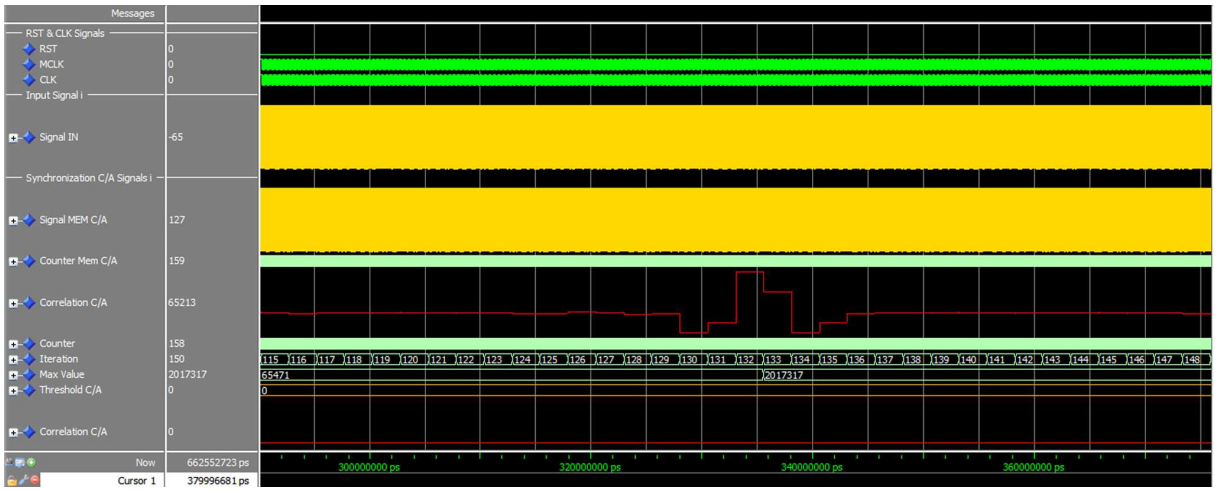
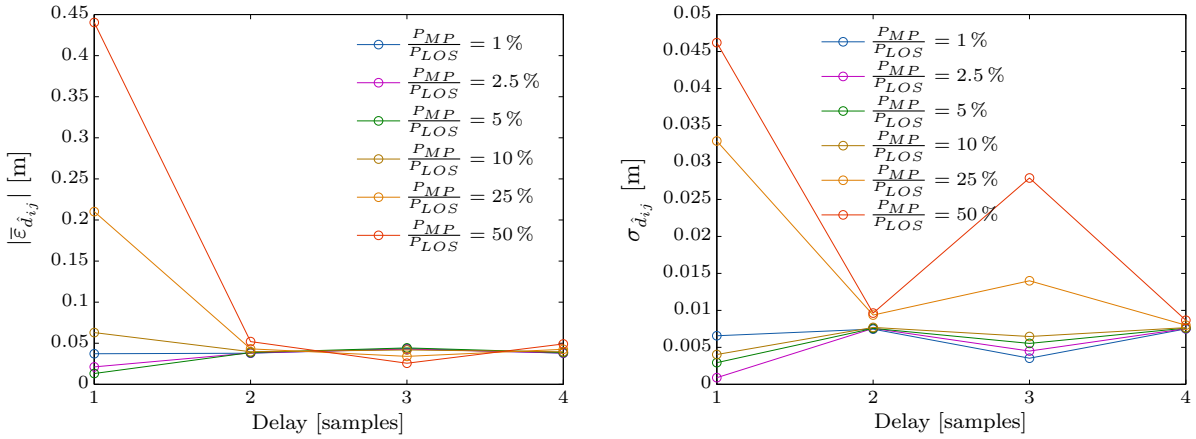


Figura 4.10: Cronograma temporal de la estimación diferencial de distancia con presencia de multicamino.

Las Figuras 4.11(a) y 4.11(b) muestran la variación del error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia para distintas potencias y retardos de la componente de multicamino. Puede verse que el multicamino más crítico, independientemente de su potencia, es

el multicamino con una muestra de retardo ($\varphi_{MP} = \varphi_{LOS} + 90^{o}$), el cual introduce un error medio de casi 0.5 m ($P_{MP}/P_{LOS} = 50\,\%$). Se observa además que el multicamino de 3 muestras de retardo ($\varphi_{MP} = \varphi_{LOS} - 90^{o}$) presenta la máxima desviación típica. En cambio, para multicaminos en fase o contrafase con la componente LOS la desviación típica obtenida es inferior a 10 mm.



(a) Error medio.

(b) Desviación típica.

Figura 4.11: Error medio y desviación típica de la estimación diferencial de distancia con presencia del multicamino.

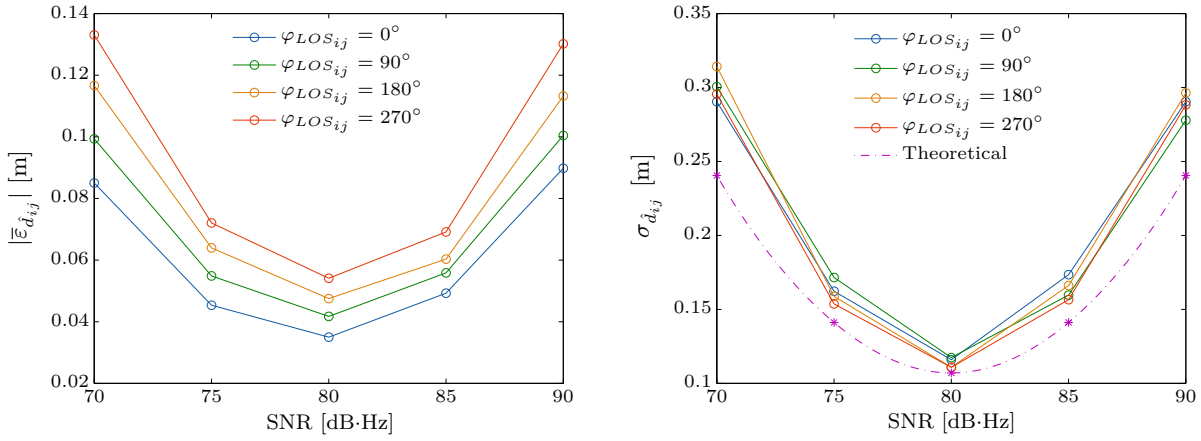
En un estudio previo a este trabajo se ha desarrollado un modelo de multicamino para el entorno CBL. En dicho estudio se asume que la reflexión del multicamino sobre las superficies del entorno es especular, y que éstas presentan una reflectividad del 50 %. Se asume además que la diferencia máxima entre el retardo de multicamino y el retardo directo es $\tau_{MP} - \tau_{LOS} \simeq 0.35 \cdot T_{chip}$, por lo que la relación de potencias ópticas entre ambas componentes es $P_{MP}/P_{LOS} \simeq 13\,\%$. Dado que las señales de entrada al sistema son eléctricas, por la conversión lineal entre potencias óptica y corriente eléctrica realizada por el fotodiodo, la relación de potencias entre las componentes LOS y MP es $P_{MP}/P_{LOS} \simeq 0.13^2 \simeq 2\,\%$. En una situación real, en el entorno de la CBL pueden producirse multicaminos de una potencia máxima del 2 % y con un retardo de hasta 2 muestras ($0.5 \cdot T_{chip}$), por lo que el error medio y la desviación típica de las medidas de distancia diferenciales son inferiores a 50 mm y a 10 mm respectivamente.

4.5 Ruido y retardo dinámico en las señales de entrada

Una vez caracterizados individualmente los efectos de la dinámica del robot móvil y del ruido (añadido principalmente por la electrónica de acondicionamiento de los receptores), se han combinado ambos efectos para ver su influencia sobre la estimación diferencial de distancia. Las medidas han sido realizadas para las mismas posiciones del robot móvil que en la prueba del apartado 4.3.2, y con la misma variación del retardo que se realizó la sección 4.2: primero en una señal, y luego en las dos señales.

4.5.1 Retardo dinámico en una única señal de entrada

En las Figuras 4.12(a) y 4.12(b) se representa el error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia para distintas posiciones del robot móvil entre dos receptores cuando además la señal que alcanza el receptor R_i (ver Figura 4.8) presenta un retardo variable. Existe una clara similitud entre los resultados obtenidos en esta prueba y los obtenidos en la prueba del apartado 4.3.2 en cuanto al perfil de evolución del error medio y la desviación típica. Sin embargo, cuando el robot móvil se encuentra bajo uno de los receptores el error medio aumenta en más de 35 mm para cualquiera de las fases en comparación con las medidas estáticas, pero a medida que se aproxima al centro, las medidas estáticas y dinámicas tienden a asemejarse; en el caso de medidas dinámicas la fase diferencial que mayor error medio presenta es $\varphi_{LOS_{ij}} = 270^{\text{o}}$, y no $\varphi_{LOS_{ij}} = 180^{\text{o}}$ como sucedía en las medidas dinámicas.



(a) Error medio.

(b) Desviación típica.

Figura 4.12: Error medio y desviación típica de la estimación diferencial de distancia con ruido en ambos receptores y retardo dinámico en una única señal.

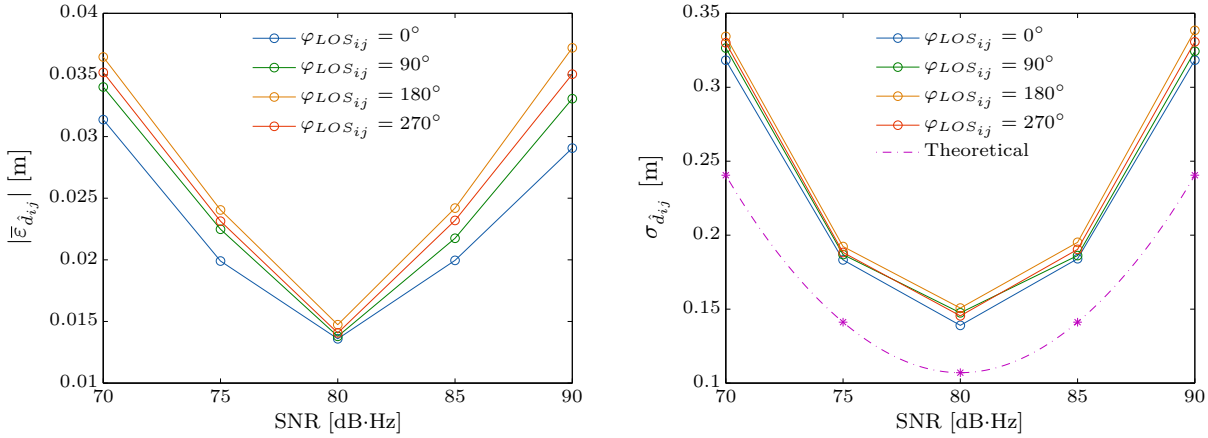
La desviación típica en las medidas dinámicas (líneas sólidas) presenta mayor variabilidad entre las distintas fases diferenciales para un SNR dado respecto a las medidas estáticas. Se observa también que cuando el robot móvil está debajo de alguno de los receptores la desviación típica sufre un ligero aumento (~ 2 mm), pero a medida que el SNR de ambas señales se iguala, la desviación se aproxima mucho más a la desviación típica teórica (línea discontinua) que en las medidas estáticas.

4.5.2 Retardo dinámico en ambas señales de entrada

Las Figuras 4.13(a) y 4.13(b) muestran el error medio absoluto ($|\overline{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la medida diferencial de distancia para distintas posiciones del robot móvil entre dos receptores cuando además las señales que alcanzan cada uno de ellos experimentan el mismo retardo dinámico. En cuanto

al error medio, se puede ver una fuerte reducción en comparación a las medidas realizadas en el apartado 4.5.1, así como una menor variabilidad entre las distintas fases diferenciales. Como ya se comentó en el apartado 4.2.2, el error medio entre las actualizaciones del retardo tiende a compensarse, y por tanto, a minimizarse, y en este caso, a pesar del ruido introducido por la electrónica de acondicionamiento, los valores de error medio obtenidos se han reducido en más de un 50 % respecto a los resultados vistos en el apartado 4.3.2.

En cambio, la desviación típica (líneas sólidas) aumenta en ~ 50 mm alejándose de la desviación teórica (línea discontinua) respecto a la medida en el apartado 4.5.1, pero la variabilidad entre las distintas fases diferenciales disminuye, aunque sigue siendo superior que en las medidas estáticas.



(a) Error medio.

(b) Desviación típica.

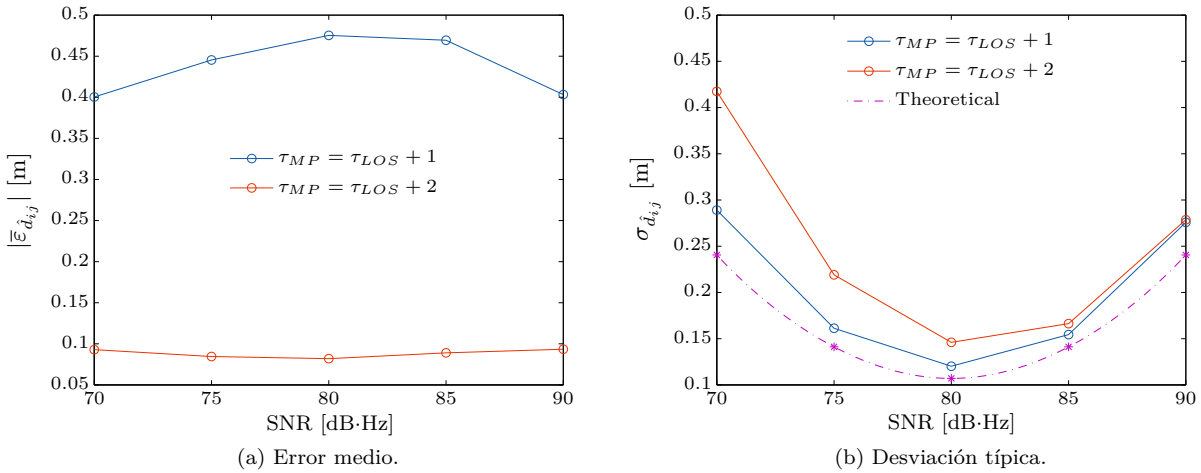
Figura 4.13: Error medio y desviación típica de la estimación diferencial de distancia con ruido en ambos receptores y retardo dinámico en ambas señales.

4.6 Ruido y presencia de multicamino en las señales de entrada

Esta prueba tiene como objetivo analizar el efecto conjunto de las dos principales fuentes de error del sistema: ruido en tensión e interferencias por multicamino. Para ello el robot móvil se ha colocado en las mismas posiciones que en la prueba del apartado 4.3.2, y además, a la señal que alcanza el receptor R_i (ver Figura 4.8) se le ha añadido una componente de multicamino con una potencia del 50 % y un retardo de 1 y 2 muestras respecto a la componente directa, que corresponden a multicaminos en cuadratura y en contrafase respectivamente.

Las Figuras 4.14(a) y 4.14(b) representan el error medio absoluto ($|\bar{\varepsilon}_{\hat{d}_{ij}}|$) y la desviación típica ($\sigma_{\hat{d}_{ij}}$) de la estimación de distancia diferencial para distintas posiciones del robot móvil entre dos receptores cuando también uno de ellos capta una componente de multicamino. Como se puede observar, el multicamino en cuadratura produce un error medio entre 0.4 - 0.5 m en función del SNR, y cuyo perfil de evolución es inverso al visto en otras pruebas con ruido; ello parece indicar que los errores de ruido y multicamino son de signo contrario. En cambio, el error medio medido en el caso del multicamino en contrafase varía entre 0.08 - 0.1 m, manteniéndose más o menos constante ya que se ve poco afectado por el ruido. Ambos valores de error medio concuerdan con los resultados vistos en la sección 4.4. En cuanto a la desviación típica, el multicamino en contrafase tiene mucha mayor influencia que el multicamino en cuadratura a medida que el SNR disminuye. De hecho, el multicamino en cuadratura se mantiene muy próximo (< 50 mm) del comportamiento teórico (línea discontinua) al igual que en las pruebas con ruido en ausencia de multicamino.

A la vista de los resultados obtenidos en esta prueba se concluye que el error medio y la desviación típica están más afectados por multicaminos en cuadratura y en contrafase respectivamente. La interferencia entre la componente LOS y MP en cuadratura altera la fase de la señal de entrada, mientras que la interferencia entre la componente LOS y MP en contrafase modifica la amplitud pero no la fase de la señal de entrada. El error medio, que es más sensible a la fase de la señal de entrada, se ve más afectado por el multicamino en cuadratura; en cambio la desviación típica, que depende mucho más de la amplitud de la señal de entrada, está más afectada por el multicamino en cuadratura.



(a) Error medio.

(b) Desviación típica.

Figura 4.14: Error medio y desviación típica de la estimación diferencial de distancia con ruido en ambos receptores y presencia del multicamino.

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

En este trabajo se ha desarrollado una propuesta de implementación en hardware digital para un sistema de medida de distancias diferenciales por IR con mitigación de multicamino. Ello ha implicado:

- Seleccionar una plataforma hardware sobre la cual llevar a cabo la implementación. Tras analizar el rendimiento y coste de varios dispositivos HW se ha escogido una FPGA en base a que proporciona un rendimiento que se ajusta al requerido por el sistema y su coste económico resulta asequible.
- Diseño e implementación de la arquitectura. Mediante el lenguaje de descripción hardware VHDL se han configurado los recursos lógicos y de interconexión de la FPGA para llevar a cabo la implementación.
- Pruebas funcionales para verificar el correcto funcionamiento de cada uno de los bloques que componen la arquitectura, así como de la arquitectura al completo.
- Pruebas temporales en condiciones no ideales (dinámica del robot móvil, ruido, e interferencias por multicamino) para validar su comportamiento en un entorno real.

El sistema es capaz de funcionar correctamente bajo las restricciones temporales impuestas, esto es, funcionar a una frecuencia de 100 MHz. La implementación a una frecuencia tan elevada ha requerido una cuidadosa, óptima, y eficiente descripción hardware del diseño. No obstante, algunos de los recursos de la FPGA (bloques de memoria RAM) han limitado alcanzar frecuencias de trabajo superiores debido a su elevado retardo intrínseco.

Los resultados, hasta donde se puede analizar, son coherentes con lo esperado teóricamente, aunque requieren un estudio más profundo en algunos aspectos (error medio y fases críticas). En cuanto a los resultados obtenidos, cabe destacar:

- Dinámica del robot móvil: la etapa de tracking ELDLL funciona correctamente llevando a cabo las actualizaciones de la estimación del retardo para $\Delta\tau < 1.984$ ms/s. En situaciones en las que ambas señales de entrada presenten un retardo dinámico el error medio se reduce ($> 90\,\%$), ya que éste tiende a compensarse entre las actualizaciones del retardo, mientras que la desviación típica aumenta en más de 5 veces respecto a las condiciones ideales.

- Ruido: se ha comprobado que el comportamiento del sistema, teniendo en cuenta el ruido generado por la electrónica de acondicionamiento de los sensores, se encuentra muy próximo al comportamiento teórico esperado en cuanto a desviación típica. El ruido (AWGN) es de media nula; sin embargo, la duración de las medidas realizadas no es lo suficientemente elevada como para realizar una estimación estadística fiable del valor medio de las medidas. Ello se traduce en un incremento del error medio en función del SNR.

- Interferencias por multicamino: el sistema ha demostrado una buena capacidad en la mitigación de multicaminos con una potencia inferior al 10 % respecto a la componente directa independientemente del retardo del multicamino. Dentro del entorno de la CBL los multicaminos más severos que pueden producirse presentan una potencia $\sim 2\,\%$ y un retardo de 2 muestras. Para estos casos el sistema ha conseguido un error medio inferior a 50 mm y una desviación típica inferior a 10 mm.

Una vez se solventen algunos de los aspectos propuestos como trabajos futuros en la sección 5.2 (etapa de digitalización, umbrales dinámicos, y resolución de las señales locales), el trabajo desarrollado sirve como núcleo del prototipo sobre el cual efectuar pruebas reales del sistema de medida.

5.2 Trabajos futuros

La implementación hardware del sistema desarrollada en este trabajo parte de señales de entrada sintéticas que ya se encuentran digitalizadas. Por este motivo, se propone como principal trabajo futuro a corto plazo el diseño e implementación de una etapa de digitalización y acondicionamiento de señal previa a la arquitectura implementada. Esta etapa se basará en un controlador para un ADC multicanal y de adquisición simultánea que gestione la comunicación entre el ADC y la arquitectura de medida y que proporcione a ésta las señales captadas en los receptores del entorno en un formato adecuado. La validación de esta etapa se realizará a partir de las señales que lleguen a los receptores y sean digitalizadas por el ADC ya conectado a la FPGA evaluándose, al igual que en este trabajo, el error medio y desviación típica de las medidas.

Del mismo modo, se plantea también el diseño e implementación de una etapa posterior encargada de suministrar al subsistema de estimación del LPS las medidas obtenidas por el sistema desarrollado, lo cual implica la comunicación con otro sistema.

Otro aspecto a mejorar directamente sobre la implementación propuesta consiste en el diseño de umbrales dinámicos para realizar la detección de pico de la función de correlación en la etapa de sincronización C/A y para actualizar la estimación continua del retardo en la etapa de tracking ELDLL. Con ello se pretende que dichos umbrales sean auto-ajustables en función de la potencia de las señales de entrada al sistema.

Con el objetivo de reducir la limitación de precisión que provoca la discretización de las funciones de correlación, se propone el aumento de la resolución de las señales locales. Para señales de entrada con n muestras/chip, si las señales locales están sobremuestradas a $x \cdot n$ muestras/chip, se consigue un aumento de x veces en la resolución de la estimación del retardo, aunque de las memorias solamente se extraigan n muestras/chip.

Las actuales características del enlace óptico hacen que el nivel de SNR medido a la salida de los receptores se encuentren dentro del rango [70 - 90] dB·Hz. Por tanto, se requiere mejorar las características del enlace IR (potencia, cobertura, ancho de banda, etc) a fin de proporcionar un rango de SNR con límites superiores que maximice la precisión en las medidas diferenciales de distancia. Ello supone por un lado la búsqueda de nuevos dispositivos fotónicos con características optimizadas, y por otro lado una optimización de la electrónica de acondicionamiento.

En el campo del posicionamiento, los recursos y frecuencias necesarios para implementar en tiempo real un LPS basado en infrarrojos se hallan en el límite tecnológico actual por lo que será necesario desarrollar alternativas que exploren características como el grado de paralelismo, la reutilización de recursos, viabilidad de implementación, etc. En base a ello se propone una línea de investigación que persiga los siguientes objetivos:

- Análisis y mejora del método de estimación de distancias diferenciales teniendo en cuenta los compromisos del sistema, y centrándose en la mitigación de los errores debidos a interferencias por multicamino. Para ello se estudiarán tanto la estructura del sistema de procesado de señal, como las características y técnica de acceso al medio de dicha señal.

- Definición y estudio de arquitecturas de procesado eficientes. La implementación real del LPS se encuentra próxima al límite tecnológico actual, por lo que se requiere el análisis de arquitecturas eficientes que permitan aplicar en tiempo real los sistemas de procesado definidos en el objetivo anterior.

Bibliografía

[1] E. Martín Gorostiza, J. L. Lázaro Galilea, F. J. Meca Meca, D. Salido Monzú, F. Espinosa Zapata, and L. Pallarés Puerto, "Infrared Sensor System for Mobile-Robot Positioning in Intelligent Spaces," *Sensors*, vol. 11, no. 12, pp. 5416–5438, May 2011. [Online]. Available: http://dx.doi.org/10.3390/s110505416

[2] A. J. Van Dierendonck, P. Fenton, and T. Ford, "Theory and Performance of Narrow Correlator Spacing in a GPS Receiver," *The Institute of Navigation*, vol. 39, no. 3, pp. 1–20, 1992. [Online]. Available: http://www.sokkia.com.tw/novatel/Documents/Papers/File2.pdf

[3] L. Garin, F. van Diggelen, and J.-M. Rousseau, "Strobe & Edge Correlator Multipath Mitigation for Code," *Proceedings of the 9th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1996)*, pp. 657–664, September 1996.

[4] V. A. Veitsel, A. V. Zhdanov, and M. I. Zhodzishsky, "The Mitigation of Multipath Errors by Strobe Correlators in GPS/GLONASS Receivers," *GPS Solutions*, vol. 2, no. 2, pp. 38–45, Oct 1998. [Online]. Available: http://dx.doi.org/10.1007/PL00000035

[5] S. M. Kanekal and M. S. Braasch, "Multipath mitigation with gated signal technology," *Proceedings of the 54th Annual Meeting of the Institute of Navigation*, no. 5, pp. 535–542, June 1998.

[6] G. A. McGraw and M. S. Braasch, "GNSS Multipath Mitigation Using Gated and High Resolution Correlator Concepts," *Proceedings of the 1999 International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1999)*, pp. 333–342, January 1999. [Online]. Available: http://www.docin.com/p-307755601.html

[7] R. D. J. van Nee, "The Multipath Estimating Delay Lock Loop," *IEEE Second International Symposium on Spread Spectrum Techniques and Applications (ISSSTA'92)*, pp. 39–42, November 1992.

[8] L. R. Weill, "Multipath Mitigation using Modernized GPS Signals: How Good Can it Get?" *Proceedings of the 15th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2002)*, pp. 493–505, September 2002.

[9] P. C. Fenton and J. Jones, "The Theory and Performance of NovAtel Inc.'s Vision Correlator," *Proceedings of the 18th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2005)*, pp. 2178–2186, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.325.4521&rep=rep1&type=pdf

[10] X. Chen, F. Dovis, and P. Mulassano, "Turbo Architecture for Multipath Mitigation in Global Navigation Satellite System Receivers," *IET Radar, Sonar & Navigation*, vol. 5, no. 5, pp. 517–527, May 2011. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5767059

[11] X. Chen and F. Dovis, "Enhanced CADLL structure for multipath mitigation in urban scenarios," *Proceedings of the 2011 International Technical Meeting of The Institute of Navigation (ION ITM 2011)*, pp. 678–686, January 2011. [Online]. Available: http://www.ion.org/publications/abstract.cfm?articleID=9512

[12] P. C. Fenton, B. Falkenberg, T. J. Ford, K. K. Ng, and A. J. V. Dierendonck, "NovAtel's GPS Receiver - The High Performance OEM Sensor of the Future," *Proceedings of the 4th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1991)*, pp. 49–58, 1991. [Online]. Available: http://www.novatel.com/assets/Documents/Papers/File1.pdf

[13] A. J. Van Dierendonck and M. Braasch, "Evaluation of GNSS Receiver Correlation Processing Techiques for Multipath and Noise Mitigation," *Proceedings of the 1997 National Technical Meeting of The Institute of Navigation*, pp. 207–215, January 1997. [Online]. Available: http://www.ion.org/publications/abstract.cfm?articleID=481

[14] F. Nunes, F. Sousa, and J. Leitao, "Gating Functions for Multipath Mitigation in GNSS BOC Signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 3, pp. 951–964, July 2007. [Online]. Available: http://dx.doi.org/10.1109/TAES.2007.4383585

[15] G. Zhang, O. H. Woon, W. Wang, S. Krishnan, F. Chin, and K. C. Chung, "Time-frequency domain delay estimation algorithm for multipath and interference mitigation in UWB positioning systems," *2009 IEEE International Conference on Ultra-Wideband*, pp. 698–701, September 2009. [Online]. Available: http://dx.doi.org/10.1109/ICUWB.2009.5288757

[16] S. Wu, Y. Ma, Q. Zhang, and N. Zhang, "NLOS Error Mitigation for UWB Ranging in Dense Multipath Environments," *2007 IEEE Wireless Communications and Networking Conference*, pp. 1565–1570, March 2007. [Online]. Available: http://dx.doi.org/10.1109/WCNC.2007.295

[17] D. A. Shaddock, "Digitally enhanced heterodyne interferometry," *Optics Letters*, vol. 32, no. 22, pp. 3355–3357, 2007. [Online]. Available: http://dx.doi.org/10.1364/OL.32.003355

[18] O. P. Lay, S. Dubovitsky, D. A. Shaddock, and B. Ware, "Coherent range-gated laser displacement metrology with compact optical head," *Optics Letters*, vol. 32, no. 20, pp. 2933–2935, 2007. [Online]. Available: http://dx.doi.org/10.1364/OL.32.002933

[19] D. Salido Monzú, A. Wieser, E. Martín Gorostiza, J. L. Lázaro Galilea, and F. Domingo Pérez, "Multipath Mitigation for a Phase-Based Infrared Ranging System Applied to Indoor Positioning," *Proceedings of IPIN 2013: International Conference on Indoor Positioning and Indoor Navigation*, October 2013.

Apéndice A

Manual de usuario

A.1 Códigos fuente

Código fuente A.1: DDMS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

library UNISIM;
use UNISIM.vcomponents.all;

----------------------------------------------------------------------------
-- DIFFERENTIAL DISTANCE MEASURING SYSTEM
----------------------------------------------------------------------------
entity DDMS is

  port (
    rst           : in  std_logic;      -- rst signal
    clk           : in  std_logic;      -- clk signal
    signal_IN_I   : in  std_logic_vector(bits_adc-1 downto 0);   -- input signal i
    signal_IN_J   : in  std_logic_vector(bits_adc-1 downto 0);   -- input signal j
    distance_diff : out std_logic_vector(bits_dist-1 downto 0)); -- differential distance estimation

end entity DDMS;


architecture rtl of DDMS is

  ----------------------------------------------------------------------------
  -- Components Declaration
  ----------------------------------------------------------------------------
  component PMS is
    port (
      rst       : in  std_logic;        -- rst signal
      clk       : in  std_logic;        -- clk signal
      signal_IN : in  std_logic_vector(bits_adc-1 downto 0);    -- input signal
      phase     : out std_logic_vector(bits_phase-1 downto 0)); -- phase estimated
  end component;

  component diff_phase is
    port (
      rst        : in  std_logic;       -- reset signal
      clk        : in  std_logic;       -- clock signal
      phase_i    : in  std_logic_vector(bits_phase-1 downto 0);  -- signal i phase
      phase_j    : in  std_logic_vector(bits_phase-1 downto 0);  -- signal j phase
```

```
      phase_diff : out std_logic_vector(bits_phase-1 downto 0)); -- differential phase
  end component;

  component phase2distance is
    port (
      rst      : in  std_logic;          -- reset signal
      clk      : in  std_logic;          -- clk signal
      phase    : in  std_logic_vector(bits_phase-1 downto 0);  -- phase estimation
      distance : out std_logic_vector(bits_dist-1 downto 0));  -- distance estimation
  end component;


  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal clk_IN : std_logic;
  signal clk_fb : std_logic;
  signal mclk   : std_logic;

  signal phase_i  : std_logic_vector(bits_phase-1 downto 0);
  signal phase_j  : std_logic_vector(bits_phase-1 downto 0);
  signal phase_ij : std_logic_vector(bits_phase-1 downto 0);


begin  -- architecture rtl

  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  -- IBUFG: Single-ended global clock input buffer (Spartan-6)
  IBUFG_inst : IBUFG
    generic map (
      IBUF_LOW_PWR => false,  -- Low power (TRUE) vs. performance (FALSE) referenced I/O standards
      IOSTANDARD   => "DEFAULT")
    port map (
      O => clk_IN,                      -- Clock buffer output
      I => clk);  -- Clock buffer input (connect directly to top-level port)
  -- End of IBUFG_inst instantiation

  -- BUFG: Global Clock Buffer (SPARTAN 6)
  BUFG_inst : BUFG
    port map (
      O => mclk,                        -- 1-bit Clock buffer output
      I => clk_fb);                     -- 1-bit Clock buffer input
  -- End of BUFG_inst instantiation

  -- DCM_SP: Digital Clock Manager (Spartan-6)
  DCM_SP_inst : DCM_SP
    generic map (
      CLKDV_DIVIDE          => 2.0,     -- CLKDV divide value (1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,...,16).
      CLKFX_DIVIDE          => 1,       -- Divide value - D - (1-32)
      CLKFX_MULTIPLY        => 4,       -- Multiply value - M - (2-32)
      CLKIN_DIVIDE_BY_2     => false,   -- CLKIN divide by two (TRUE/FALSE)
      CLKIN_PERIOD          => 10.0,    -- Input clock period specified in nS
      CLKOUT_PHASE_SHIFT    => "NONE",  -- Output phase shift (NONE, FIXED, VARIABLE)
      CLK_FEEDBACK          => "1X",    -- Feedback source (NONE, 1X, 2X)
      DESKEW_ADJUST         => "SYSTEM_SYNCHRONOUS",  -- SYSTEM_SYNCHRNOUS or SOURCE_SYNCHRONOUS
      DFS_FREQUENCY_MODE    => "LOW",   -- Unsupported - Do not change value
      DLL_FREQUENCY_MODE    => "LOW",   -- Unsupported - Do not change value
      DSS_MODE              => "NONE",  -- Unsupported - Do not change value
      DUTY_CYCLE_CORRECTION => true,    -- Unsupported - Do not change value
      FACTORY_JF            => X"c080", -- Unsupported - Do not change value
      PHASE_SHIFT           => 0,       -- Amount of fixed phase shift (-255 to 255)
      STARTUP_WAIT          => false)   -- Delay config DONE until DCM LOCKED (TRUE/FALSE)

    port map (
      CLK0     => clk_fb,               -- 1-bit 0 degree clock output
      CLK180   => open,                 -- 1-bit 180 degree clock output
      CLK270   => open,                 -- 1-bit 270 degree clock output
      CLK2X    => open,                 -- 1-bit 2X clock frequency clock output
```

```
      CLK2X180 => open,                -- 1-bit 2X clock frequency, 180 degree clock output
      CLK90    => open,                -- 1-bit 90 degree clock output
      CLKDV    => open,                -- 1-bit Divided clock output
      CLKFX    => open,                -- 1-bit Digital Frequency Synthesizer output (DFS)
      CLKFX180 => open,                -- 1-bit 180 degree CLKFX output
      LOCKED   => open,                -- 1-bit DCM Lock Output
      PSDONE   => open,                -- 1-bit Phase shift done output
      STATUS   => open,                -- 8-bit DCM status output
      CLKFB    => mclk,                -- 1-bit Clock feedback input
      CLKIN    => clk_IN,              -- 1-bit Clock input
      DSSEN    => '0',                 -- 1-bit Unsupported
      PSCLK    => '0',                 -- 1-bit Phase shift clock input
      PSEN     => '0',                 -- 1-bit Phase shift enable
      PSINCDEC => '0',                 -- 1-bit Phase shift increment/decrement input
      RST      => '0');                -- 1-bit Active high reset input
  -- End of DCM_SP_inst instantiation

  PMS_i_inst : PMS
    port map (
      rst       => rst,
      clk       => mclk,
      signal_IN => signal_IN_i,
      phase     => phase_i);

  PMS_j_inst : PMS
    port map (
      rst       => rst,
      clk       => mclk,
      signal_IN => signal_IN_j,
      phase     => phase_j);

  diff_phase_inst : diff_phase
    port map (
      rst        => rst,
      clk        => mclk,
      phase_i    => phase_i,
      phase_j    => phase_j,
      phase_diff => phase_ij);

  phase2distance_inst : phase2distance
    port map (
      rst      => rst,
      clk      => mclk,
      phase    => phase_ij,
      distance => distance_diff);

end architecture rtl;
```

Código fuente A.2: PMS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- PHASE MEASURING SYSTEM
-------------------------------------------------------------------------------
entity PMS is

  port (
    rst       : in  std_logic;          -- rst signal
    clk       : in  std_logic;          -- clk signal
    signal_IN : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
    phase     : out std_logic_vector(bits_phase-1 downto 0));  -- phase estimated

end entity PMS;
```

```
architecture rtl of PMS is

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  component synchronization_CA is
    port (
      rst       : in  std_logic;        -- rst signal
      clk       : in  std_logic;        -- clk signal
      signal_IN : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
      ready_CA  : out std_logic;                                  -- C/A synchronization ready
      delay_CA  : out std_logic_vector(bits_samples-1 downto 0)); -- C/A delay estimation
  end component;

  component tracking_ELDLL is
    port (
      rst       : in  std_logic;        -- rst signal
      clk       : in  std_logic;        -- clk signal
      enable    : in  std_logic;        -- enable tracking stage
      signal_IN : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
      delay_CA  : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay estimation
      delay     : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation
  end component;

  component demodulation_DSSS is
    port (
      rst         : in  std_logic;      -- rst signal
      clk         : in  std_logic;      -- clk signal
      enable      : in  std_logic;      -- enable DSSS demodulation stage
      signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
      delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
      ready_DSSS  : out std_logic;                                 -- DSSS demodulation ready
      signal_DSSS : out std_logic_vector(bits_adc-1 downto 0));    -- DSSS demodulated input signal
  end component;

  component phase_estimation is
    port (
      rst       : in  std_logic;        -- rst signal
      clk       : in  std_logic;        -- clk signal
      enable    : in  std_logic;        -- enable phase estimation stage
      signal_IN : in  std_logic_vector(bits_adc-1 downto 0);    -- input signal
      phase     : out std_logic_vector(bits_phase-1 downto 0)); -- phase estimated
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal ready_CA_i    : std_logic;
  signal delay_CA_i    : std_logic_vector(bits_samples-1 downto 0);
  signal delay_i       : std_logic_vector(bits_samples-1 downto 0);
  signal ready_DSSS_i  : std_logic;
  signal signal_DSSS_i : std_logic_vector(bits_adc-1 downto 0);


begin  -- architecture rtl

  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  synchronization_CA_inst : synchronization_CA
    port map (
      rst       => rst,
      clk       => clk,
      signal_IN => signal_IN,
      ready_CA  => ready_CA_i,
      delay_CA  => delay_CA_i);

  tracking_ELDLL_inst : tracking_ELDLL
```

```
    port map (
      rst       => rst,
      clk       => clk,
      enable    => ready_CA_i,
      signal_IN => signal_IN,
      delay_CA  => delay_CA_i,
      delay     => delay_i);

  demodulation_DSSS_inst : demodulation_DSSS
    port map (
      rst         => rst,
      clk         => clk,
      enable      => ready_CA_i,
      signal_IN   => signal_IN,
      delay       => delay_i,
      ready_DSSS  => ready_DSSS_i,
      signal_DSSS => signal_DSSS_i);

  phase_estimation_inst : phase_estimation
    port map (
      rst       => rst,
      clk       => clk,
      enable    => ready_DSSS_i,
      signal_IN => signal_DSSS_i,
      phase     => phase);

end architecture rtl;
```

Código fuente A.3: synchronization_CA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- COARSE/ACQUISITION SYNCHRONIZATION STAGE
-------------------------------------------------------------------------------
entity synchronization_CA is

  port (
    rst       : in  std_logic;          -- rst signal
    clk       : in  std_logic;          -- clk signal
    signal_IN : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
    ready_CA  : out std_logic;                                  -- C/A Synchronization ready
    delay_CA  : out std_logic_vector(bits_samples-1 downto 0)); -- C/A delay estimation

end entity synchronization_CA;


architecture rtl of synchronization_CA is

  -------------------------------------------------------------------------------
  -- Components Declaration
  -------------------------------------------------------------------------------
  component correlator_CA is
    port (
      rst               : in  std_logic;  -- rst signal
      clk               : in  std_logic;  -- clk signal
      enable            : in  std_logic;  -- enable correlation signal
      signal_IN         : in  std_logic_vector(bits_adc-1 downto 0);   -- input signal
      correlation_ready : out std_logic;  -- C/A correlation value ready
      correlation       : out std_logic_vector(bits_corr-1 downto 0);  -- C/A correlation values
      threshold         : out std_logic_vector(bits_corr-1 downto 0)); -- C/A threshold
  end component;

  component buffer_CA is
    port (
```

```
      rst        : in  std_logic;       -- rst signal
      clk        : in  std_logic;       -- clk signal
      enable     : in  std_logic;       -- enable signal
      sample_IN  : in  std_logic_vector(bits_corr-1 downto 0);  -- input sample
      mem_full   : out std_logic;       -- memory full
      sample_OUT : out std_logic_vector(bits_corr-1 downto 0)); -- output sample
  end component;

  component peak_detection is
    port (
      rst         : in  std_logic;      -- reset signal
      clk         : in  std_logic;      -- clock signal
      enable      : in  std_logic;      -- enable detection
      correlation : in  std_logic_vector(bits_corr-1 downto 0);    -- C/A correlation values
      threshold   : in  std_logic_vector(bits_corr-1 downto 0);    -- C/A threshold value
      delay       : out std_logic_vector(bits_samples-1 downto 0)  -- C/A delay estimation
      ready       : out std_logic);     -- C/A synchronization ready
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal signal_IN_1          : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_2          : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_CA         : std_logic_vector(bits_adc-1 downto 0);
  signal mem_ready            : std_logic;
  signal correlation_CA_ready : std_logic;
  signal correlation_CA       : std_logic_vector(bits_corr-1 downto 0);
  signal correlation_CA_i     : std_logic_vector(bits_corr-1 downto 0);
  signal threshold_CA         : std_logic_vector(bits_corr-1 downto 0);


begin  -- architecture rtl

  signal_IN_CA <= signal_IN_2;

  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  correlator_CA_inst : correlator_CA
    port map (
      rst               => rst,
      clk               => clk,
      enable            => '1',
      signal_IN         => signal_IN_CA,
      correlation_ready => correlation_CA_ready,
      correlation       => correlation_CA,
      threshold         => threshold_CA);

  buffer_CA_inst : buffer_CA
    port map (
      rst        => rst,
      clk        => clk,
      enable     => correlation_CA_ready,
      sample_IN  => correlation_CA,
      mem_full   => mem_ready,
      sample_OUT => correlation_CA_i);

  peak_detection_inst : peak_detection
    port map (
      rst         => rst,
      clk         => clk,
      enable      => mem_ready,
      correlation => correlation_CA_i,
      threshold   => threshold_CA,
      delay       => delay_CA,
      ready       => ready_CA);
```

```
  ---------------------------------------------------------------------------
  -- Delay Process To Synchronize Input & Memory Signals (2 MEM Latency Cycles)
  ---------------------------------------------------------------------------
  delaying : process (clk, rst) is
  begin  -- process delay

    if rst = '1' then                   -- asynchronous reset (active high)

      signal_IN_1 <= (others => '0');
      signal_IN_2 <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      signal_IN_1 <= signal_IN;         -- Latency from C/A Synchronization Stage
      signal_IN_2 <= signal_IN_1;

    end if;

  end process delaying;

end architecture rtl;
```

Código fuente A.4: correlator_CA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

library UNISIM;
use UNISIM.vcomponents.all;

library UNIMACRO;
use UNIMACRO.vcomponents.all;

use work.constants_DDMS.all;

---------------------------------------------------------------------------
-- CORRELATOR C/A
---------------------------------------------------------------------------
entity correlator_CA is

  port (
    rst              : in  std_logic;  -- rst signal
    clk              : in  std_logic;  -- clk signal
    enable           : in  std_logic;  -- enable correlation signal
    signal_IN        : in  std_logic_vector(bits_adc-1 downto 0);   -- input signal
    correlation_ready : out std_logic;                              -- C/A correlation value ready
    correlation      : out std_logic_vector(bits_corr-1 downto 0);  -- C/A correlation values
    threshold        : out std_logic_vector(bits_corr-1 downto 0)); -- C/A threshold

end correlator_CA;


architecture rtl of correlator_CA is

  ---------------------------------------------------------------------------
  -- Components Declaration
  ---------------------------------------------------------------------------
  component buffer_MEM_CA
    port (
      rst    : in  std_logic;                              -- rst signal
      clk    : in  std_logic;                              -- clk signal
      enable : in  std_logic;                              -- enable signal
      sample : out std_logic_vector(bits_adc-1 downto 0));  -- output sample
  end component;
```

```
  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal signal_MEM          : std_logic_vector(bits_adc-1 downto 0);
  signal result              : std_logic_vector(bits_corr-1 downto 0);
  signal correlation_i       : std_logic_vector(bits_corr-1 downto 0);
  signal correlation_ready_i : std_logic;
  signal sclr_macc           : std_logic;
  signal max_value           : std_logic_vector(bits_corr-1 downto 0);
  signal counter             : std_logic_vector(bits_samples downto 0);
  signal iteration           : std_logic_vector(bits_samples downto 0);


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  correlation_ready <= correlation_ready_i;
  correlation       <= correlation_i;


  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  buffer_MEM_CA_inst : buffer_MEM_CA
    port map (
      rst    => rst,
      clk    => clk,
      enable => enable,
      sample => signal_MEM);

  -- MACC_MACRO: Multiple Accumulate Function implemented in a DSP48E (Spartan 6)
  MACC_MACRO_inst : MACC_MACRO
    generic map (
      DEVICE  => "SPARTAN6",            -- Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6"
      LATENCY => 1,                     -- Desired clock cycle latency, 1-4
      WIDTH_A => bits_adc,              -- Multiplier A-input bus width, 1-25
      WIDTH_B => bits_adc,              -- Multiplier B-input bus width, 1-18
      WIDTH_P => bits_corr)             -- Accumulator output bus width, 1-48
    port map (
      P         => result,              -- MACC ouput bus, width determined by WIDTH_P generic
      A         => signal_IN,           -- MACC input A bus, width determined by WIDTH_A generic
      ADDSUB    => '1',                 -- 1-bit add/sub input, high selects add, low selects subtract
      B         => signal_MEM,          -- MACC input B bus, width determined by WIDTH_B generic
      CARRYIN   => '0',                 -- 1-bit carry-in input to accumulator
      CE        => '1',                 -- 1-bit active high input clock enable
      CLK       => clk,                 -- 1-bit positive edge clock input
      LOAD      => '0',                 -- 1-bit active high input load accumulator enable
      LOAD_DATA => (others => '0'),     -- Load accumulator input data, width determined by WIDTH_P generic
      RST       => sclr_macc);          -- 1-bit input active high reset
  -- End of MACC_MACRO_inst instantiation


  ---------------------------------------------------------------------------
  -- End of Samples Processing Process
  ---------------------------------------------------------------------------
  processing_end : process (clk, rst)
  begin  -- process processing_end

    if rst = '1' then                   -- asynchronous reset (active high)

      counter            <= (others => '0');
      iteration          <= (others => '0');
      correlation_i      <= (others => '0');
      correlation_ready_i <= '0';

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then
```

```
      counter <= counter + 1;

      if counter = n_samples-1 then

        counter             <= (others => '0');
        correlation_i       <= result;
        correlation_ready_i <= '1';
        iteration           <= iteration + 1;

      else

        correlation_ready_i <= '0';

      end if;

      if iteration = n_samples-1 and counter = n_samples-1 then

        iteration <= (others => '0');

      end if;

    end if;

  end if;

end process processing_end;


-------------------------------------------------------------------------------
-- Max Correlation Value Process
-------------------------------------------------------------------------------
max : process (clk, rst)
begin  -- max

  if rst = '1' then                   -- asynchronous reset (active high)

    max_value <= (others => '0');
    threshold <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    if counter = n_samples-1 and iteration < n_samples then

      if correlation_i > max_value then

        max_value <= correlation_i;

      end if;

    end if;

    if iteration = 0 and correlation_ready_i = '1' then

      threshold <= ('0' & max_value(bits_corr-1 downto 1)) + ("00" & max_value(bits_corr-1 downto 2));
      max_value <= (others => '0');

    end if;

  end if;

end process max;


-------------------------------------------------------------------------------
-- MACC Reset Process
-------------------------------------------------------------------------------
macc_reset : process (clk, rst)
begin  -- process macc_reset
```

```
    if rst = '1' then                   -- asynchronous reset (active high)
      sclr_macc <= '1';
    elsif clk'event and clk = '1' then  -- rising clock edge
      if counter = n_samples-1 then
        sclr_macc <= '1';
      else
        sclr_macc <= '0';
      end if;
    end if;
  end process macc_reset;
end rtl;
```

Código fuente A.5: buffer_MEM_CA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- BUFFER CORRELATION MEM C/A
-------------------------------------------------------------------------------
entity buffer_MEM_CA is

  port (
    rst    : in  std_logic;                              -- rst signal
    clk    : in  std_logic;                              -- clk signal
    enable : in  std_logic;                              -- enable signal
    sample : out std_logic_vector(bits_adc-1 downto 0));  -- output sample

end buffer_MEM_CA;


architecture rtl of buffer_MEM_CA is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component memory_local
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples downto 0);
      data_out : out std_logic_vector(bits_adc-1 downto 0));
  end component;


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal address   : std_logic_vector(bits_samples downto 0);
  signal counter   : std_logic_vector(bits_samples downto 0);
  signal iteration : std_logic_vector(bits_samples downto 0);


begin  -- rtl
```

```
--------------------------------------------------------------------------------
-- Signals Instantiation
--------------------------------------------------------------------------------
address <= (counter + iteration) when ((counter + iteration) < n_samples)
           else ((counter + iteration) - n_samples);


--------------------------------------------------------------------------------
-- Components Instantiation
--------------------------------------------------------------------------------
memory_local_inst : memory_local
  port map (
    rst      => rst,
    clk      => clk,
    enable   => enable,
    r_w      => '0',
    address  => address,
    data_out => sample);


--------------------------------------------------------------------------------
-- Local Signal Reading Process
--------------------------------------------------------------------------------
process (clk, rst)
begin  -- process

  if rst = '1' then                   -- asynchronous reset (active high)

    counter <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    if enable = '1' then

      if counter < n_samples-1 then

        counter <= counter + 1;

      else

        counter <= (others => '0');

      end if;

    end if;

  end if;

end process;


process (clk, rst)
begin

  if rst = '1' then                   -- asynchronous reset (active high)

    iteration <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    if counter = n_samples-1 then

      if iteration < n_samples-1 then

        iteration <= iteration + 1;

      else

        iteration <= (others => '0');
```

```
        end if;

      end if;

    end if;

  end process;

end rtl;
```

Código fuente A.6: memory_local.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- LOCAL MEMORY (BRAM)
-------------------------------------------------------------------------------
entity memory_local is

  port (
    rst      : in  std_logic;
    clk      : in  std_logic;
    enable   : in  std_logic;
    r_w      : in  std_logic;
    address  : in  std_logic_vector(bits_samples downto 0);
    data_out : out std_logic_vector(bits_adc-1 downto 0));

end memory_local;


architecture rtl of memory_local is

  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal r_w_i      : std_logic_vector(3 downto 0);
  signal address_i  : std_logic_vector(13 downto 0);
  signal data_in_i  : std_logic_vector(31 downto 0);
  signal data_out_i : std_logic_vector(31 downto 0);


begin  -- rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
  r_w_i     <= (others => r_w);
  address_i <= "00" & address & "000";
  data_out  <= data_out_i(bits_adc-1 downto 0);


  -----------------------------------------------------------------------------
  -- Components Instantiation
  -----------------------------------------------------------------------------
  -- RAMB16BWER: 16K-bit Data and 2K-bit Parity Configurable Synchronous Dual Port
  -- Block RAM with Optional Output Registers
  -- Spartan-6, Xilinx HDL Libraries Guide, version 12.1
  mem_local_inst : RAMB16BWER
    generic map (
      -- DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
      DATA_WIDTH_A        => 9,
```

```
      DATA_WIDTH_B        => 0,
      -- DOA_REG/DOB_REG: Optional output register (0 or 1)
      DOA_REG             => 1,
      DOB_REG             => 0,
      -- EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
      EN_RSTRAM_A         => true,
      EN_RSTRAM_B         => false,
      -- INIT_00 to INIT_3F: Initial memory contents: fm = fchip = 25 MHz, phi0 = 0°, fs = 100 MHz
      INIT_00             => X"7fff80007fff800080007fff7fff80007fff80007fff80007fff80007fff8000",
      INIT_01             => X"80007fff7f0080007fff80007fff800080007fff80007fff7fff80007fff8000",
      INIT_02             => X"80007fff80007fff80007fff80007fff7fff80007fff800080007fff7fff8000",
      INIT_03             => X"80007fff80007fff80007f007fff80007fff80007fff800080007fff7fff8000",
      INIT_04             => X"80ff7fff7fff800080007fff7fff80007f00800080007fff7fff80007fff8000",
      INIT_05             => X"80007f0080007fff7fff800080ff7fff80007fff80007fff7f00800080007fff",
      INIT_06             => X"80007fff7f00800080007fff7f00800080007fff80007f007fff80007fff8000",
      INIT_07             => X"0000000080007fff80007fff80007fff80007fff80007f0080007fff7fff8000",
      -- INIT_A/INIT_B: Initial values on output port
      INIT_A              => X"000000000",
      INIT_B              => X"000000000",
      -- INIT_FILE: Optional file used to specify initial RAM contents
      INIT_FILE           => "NONE",
      -- RSTTYPE: "SYNC" or "ASYNC"
      RSTTYPE             => "SYNC",
      -- RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
      RST_PRIORITY_A      => "SR",
      RST_PRIORITY_B      => "CE",
      -- SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY", "GENERATE_X_ONLY" or "NONE"
      SIM_COLLISION_CHECK => "ALL",
      -- SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
      SIM_DEVICE          => "SPARTAN6",
      -- SRVAL_A/SRVAL_B: Set/Reset value for RAM output
      SRVAL_A             => X"000000000",
      SRVAL_B             => X"000000000",
      -- WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
      WRITE_MODE_A        => "NO_CHANGE",
      WRITE_MODE_B        => "NO_CHANGE")

    port map (
      DOA    => data_out_i,             -- 32-bit A port data output
      DOB    => open,                   -- 32-bit B port data output
      DOPA   => open,                   -- 4-bit A port parity output
      DOPB   => open,                   -- 4-bit B port parity output
      ADDRA  => address_i,              -- 14-bit A port address input
      ADDRB  => "00" & x"000",          -- 14-bit B port address input
      CLKA   => clk,                    -- 1-bit A port clock input
      CLKB   => '0',                    -- 1-bit B port clock input
      DIA    => x"00000000",            -- 32-bit A port data input
      DIB    => x"00000000",            -- 32-bit B port data input
      DIPA   => "0000",                 -- 4-bit A port parity input
      DIPB   => "0000",                 -- 4-bit B port parity input
      ENA    => enable,                 -- 1-bit A port enable input
      ENB    => '0',                    -- 1-bit B port enable input
      REGCEA => '1',                    -- 1-bit A port register clock enable input
      REGCEB => '0',                    -- 1-bit B port register clock enable input
      RSTA   => rst,                    -- 1-bit A port register set/reset input
      RSTB   => '0',                    -- 1-bit B port register set/reset input
      WEA    => r_w_i,                  -- 4-bit Port A byte-wide write enable input
      WEB    => r_w_i);                 -- 4-bit Port B byte-wide write enable input
  -- End of RAMB16BWER_inst instantiation

end rtl;
```

Código fuente A.7: buffer_CA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.constants_DDMS.all;
```

```
--------------------------------------------------------------------------------
-- BUFFER C/A
--------------------------------------------------------------------------------
entity buffer_CA is

  port (
    rst        : in  std_logic;                                  -- rst signal
    clk        : in  std_logic;                                  -- clk signal
    enable     : in  std_logic;                                  -- enable signal
    sample_IN  : in  std_logic_vector(bits_corr-1 downto 0);     -- input sample
    mem_full   : out std_logic;                                  -- memory full
    sample_OUT : out std_logic_vector(bits_corr-1 downto 0));  -- output sample

end buffer_CA;


architecture rtl of buffer_CA is

  --------------------------------------------------------------------------------
  -- Components Declaration
  --------------------------------------------------------------------------------
  component memory_CA
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples+1 downto 0);
      data_in  : in  std_logic_vector(bits_corr-1 downto 0);
      data_out : out std_logic_vector(bits_corr-1 downto 0));
  end component;


  --------------------------------------------------------------------------------
  -- Signals Declaration
  --------------------------------------------------------------------------------
  signal mem_full_i : std_logic;
  signal r_w        : std_logic;
  signal address    : std_logic_vector(9 downto 0);
  signal address_r  : std_logic_vector(bits_samples+1 downto 0);
  signal address_w  : std_logic_vector(bits_samples+1 downto 0);


begin  -- rtl

  --------------------------------------------------------------------------------
  -- Signals Instantiation
  --------------------------------------------------------------------------------
  address  <= address_r when (mem_full_i = '1') else address_w;
  mem_full <= mem_full_i;


  --------------------------------------------------------------------------------
  -- Components Instantiation
  --------------------------------------------------------------------------------
  memory_CA_inst : memory_CA
    port map (
      rst      => rst,
      clk      => clk,
      enable   => '1',
      r_w      => r_w,
      address  => address,
      data_in  => sample_IN,
      data_out => sample_OUT);


  --------------------------------------------------------------------------------
  -- Writing Process
  --------------------------------------------------------------------------------
```

```
writing : process (clk, rst) is
begin  -- process writing

  if rst = '1' then                   -- asynchronous reset (active high)

    r_w        <= '1';
    mem_full_i <= '0';
    address_w  <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    if mem_full_i = '0' then

      r_w <= '1';                     -- writing

      if enable = '1' then

        address_w <= address_w + 4;

        if address_w = 4*(n_samples-1) then

          mem_full_i <= '1';
          address_w  <= "00" & X"04";
          r_w        <= '0';          -- reading

        end if;

      end if;

    else

      if address_r = 4*(n_samples-1) then

        mem_full_i <= '0';

      end if;

    end if;

  end if;

end process writing;


-------------------------------------------------------------------------------
-- Reading Process
-------------------------------------------------------------------------------
reading : process (clk, rst) is
begin  -- process reading

  if rst = '1' then                   -- asynchronous reset (active high)

    address_r <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    if mem_full_i = '1' then

      address_r <= address_r + 4;

      if address_r = 4*(n_samples-1) then

        address_r <= (others => '0');

      end if;

    end if;

  end if;
```

```
  end process reading;

end rtl;
```

Código fuente A.8: memory_CA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- MEMORY CORRELATION C/A
-------------------------------------------------------------------------------
entity memory_CA is

  port (
    rst      : in  std_logic;
    clk      : in  std_logic;
    enable   : in  std_logic;
    r_w      : in  std_logic;
    address  : in  std_logic_vector(bits_samples+1 downto 0);
    data_in  : in  std_logic_vector(bits_corr-1 downto 0);
    data_out : out std_logic_vector(bits_corr-1 downto 0));

end memory_CA;


architecture rtl of memory_CA is

  -------------------------------------------------------------------------------
  -- Signals Declaration
  -------------------------------------------------------------------------------
  signal r_w_i      : std_logic_vector(3 downto 0);
  signal address_i  : std_logic_vector(13 downto 0);
  signal data_in_i  : std_logic_vector(31 downto 0);
  signal data_out_i : std_logic_vector(31 downto 0);


begin  -- rtl

  -------------------------------------------------------------------------------
  -- Signals Instantiation
  -------------------------------------------------------------------------------
  r_w_i                          <= (others => r_w);
  address_i                      <= '0' & address & "000";
  data_in_i(bits_corr-1 downto 0) <= data_in;
  data_in_i(31 downto bits_corr)  <= (others => '0');
  data_out                       <= data_out_i(bits_corr-1 downto 0);


  -------------------------------------------------------------------------------
  -- Components Instantiation
  -------------------------------------------------------------------------------
  -- RAMB16BWER: 16K-bit Data and 2K-bit Parity Configurable Synchronous Dual Port
  -- Block RAM with Optional Output Registers
  -- Spartan-6, Xilinx HDL Libraries Guide, version 12.1
  mem_CA_inst : RAMB16BWER
    generic map (
      -- DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
      DATA_WIDTH_A        => 36,
      DATA_WIDTH_B        => 0,
      -- DOA_REG/DOB_REG: Optional output register (0 or 1)
      DOA_REG             => 1,
      DOB_REG             => 0,
```

```
      -- EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
      EN_RSTRAM_A         => true,
      EN_RSTRAM_B         => true,
      -- INIT_A/INIT_B: Initial values on output port
      INIT_A              => X"000000000",
      INIT_B              => X"000000000",
      -- INIT_FILE: Optional file used to specify initial RAM contents
      INIT_FILE           => "NONE",
      -- RSTTYPE: "SYNC" or "ASYNC"
      RSTTYPE             => "SYNC",
      -- RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
      RST_PRIORITY_A      => "SR",
      RST_PRIORITY_B      => "SR",
      -- SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY", "GENERATE_X_ONLY" or "NONE"
      SIM_COLLISION_CHECK => "ALL",
      -- SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
      SIM_DEVICE          => "SPARTAN6",
      -- SRVAL_A/SRVAL_B: Set/Reset value for RAM output
      SRVAL_A             => X"000000000",
      SRVAL_B             => X"000000000",
      -- WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
      WRITE_MODE_A        => "NO_CHANGE",
      WRITE_MODE_B        => "NO_CHANGE")

    port map (
      DOA    => data_out_i,             -- 32-bit A port data output
      DOB    => open,                   -- 32-bit B port data output
      DOPA   => open,                   -- 4-bit A port parity output
      DOPB   => open,                   -- 4-bit B port parity output
      ADDRA  => address_i,              -- 14-bit A port address input
      ADDRB  => "00" & x"000",          -- 14-bit B port address input
      CLKA   => clk,                    -- 1-bit A port clock input
      CLKB   => '0',                    -- 1-bit B port clock input
      DIA    => data_in_i,              -- 32-bit A port data input
      DIB    => x"00000000",            -- 32-bit B port data input
      DIPA   => "0000",                 -- 4-bit A port parity input
      DIPB   => "0000",                 -- 4-bit B port parity input
      ENA    => enable,                 -- 1-bit A port enable input
      ENB    => '0',                    -- 1-bit B port enable input
      REGCEA => '1',                    -- 1-bit A port register clock enable input
      REGCEB => '0',                    -- 1-bit B port register clock enable input
      RSTA   => rst,                    -- 1-bit A port register set/reset input
      RSTB   => '0',                    -- 1-bit B port register set/reset input
      WEA    => r_w_i,                  -- 4-bit Port A byte-wide write enable input
      WEB    => r_w_i);                 -- 4-bit Port B byte-wide write enable input


end rtl;
```

Código fuente A.9: peak_detection.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;


--------------------------------------------------------------------------------
-- PEAK DETECTION C/A
--------------------------------------------------------------------------------
entity peak_detection is

  port (
    rst         : in  std_logic;        -- reset signal
    clk         : in  std_logic;        -- clock signal
    enable      : in  std_logic;        -- enable detection
    correlation : in  std_logic_vector(bits_corr-1 downto 0);    -- C/A correlation values
    threshold   : in  std_logic_vector(bits_corr-1 downto 0);    -- C/A threshold value
```

```
    delay       : out std_logic_vector(bits_samples-1 downto 0); -- C/A delay estimation
    ready       : out std_logic);       -- C/A synchronization is ready

end entity peak_detection;


architecture rtl of peak_detection is

  ---------------------------------------------------------------------------
  -- SIGNALS DECLARATION
  ---------------------------------------------------------------------------
  signal index   : std_logic_vector(bits_samples downto 0);
  signal counter : std_logic_vector(bits_samples downto 0);
  signal delay_i : std_logic_vector(bits_samples downto 0);
  signal ready_i : std_logic;


begin  -- architecture rtl

  ---------------------------------------------------------------------------
  -- Peak Detection Process
  ---------------------------------------------------------------------------
  detection : process (clk, rst) is
  begin  -- process detection

    if rst = '1' then                   -- asynchronous reset (active high)

      index   <= (others => '0');
      delay_i <= (others => '0');
      ready_i <= '0';
      delay   <= (others => '0');
      ready   <= '0';

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' and ready_i = '0' then

        if index < n_samples-1 then

          index <= index + 1;

        end if;

        if correlation > threshold then

          delay_i <= index - 2;         -- to compensate memory C/A latency
          ready_i <= '1';

        end if;

      end if;

      -- To synchronize with the input signal
      if counter = 1 and ready_i = '1' then

        delay <= delay_i(bits_samples-1 downto 0);
        ready <= ready_i;

      end if;

    end if;

  end process detection;


  process (clk, rst) is
  begin  -- process

    if rst = '1' then                   -- asynchronous reset (active high)
```

```
      counter <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if counter < n_samples-1 then

        counter <= counter + 1;

      else

        counter <= (others => '0');

      end if;

    end if;

  end process;

end architecture rtl;
```

Código fuente A.10: tracking_ELDLL.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- TRACKING EARLY-LATE DLL STAGE
--------------------------------------------------------------------------------
entity tracking_ELDLL is

  port (
    rst       : in  std_logic;          -- reset signal
    clk       : in  std_logic;          -- clock signal
    enable    : in  std_logic;          -- enable tracking flag
    signal_IN : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
    delay_CA  : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay between input & local signal
    delay     : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation

end entity tracking_ELDLL;


architecture rtl of tracking_ELDLL is

  --------------------------------------------------------------------------------
  -- Components Declaration
  --------------------------------------------------------------------------------
  component correlator_EL is
    port (
      rst               : in  std_logic;  -- rst signal
      clk               : in  std_logic;  -- clk signal
      enable            : in  std_logic;  -- enable tracking signal
      signal_IN         : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
      delay_CA          : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay estimation
      update_up         : in  std_logic;  -- increase delay estimation
      update_down       : in  std_logic;  -- decrease delay estimation
      ready_EL          : out std_logic;  -- early-late correlation ready
      correlation_early : out std_logic_vector(bits_corr-1 downto 0);     -- early correlation value
      correlation_late  : out std_logic_vector(bits_corr-1 downto 0);     -- late correlation value
      delay             : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation
  end component;

  component logic_control is
    port (
      rst               : in  std_logic;   -- reset signal
      clk               : in  std_logic;   -- clock signal
      enable            : in  std_logic;   -- early-late correlation ready
```

```
      correlation_early : in  std_logic_vector(bits_corr-1 downto 0);  -- correlation early value
      correlation_late  : in  std_logic_vector(bits_corr-1 downto 0);  -- correlation late value
      update_up         : out std_logic;   -- increase delay estimation
      update_down       : out std_logic);  -- decrease delay estimation
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal delay_CA_i          : std_logic_vector(bits_samples-1 downto 0);
  signal ready_EL_i          : std_logic;
  signal correlation_early_i : std_logic_vector(bits_corr-1 downto 0);
  signal correlation_late_i  : std_logic_vector(bits_corr-1 downto 0);
  signal update_up_i         : std_logic;
  signal update_down_i       : std_logic;


begin  -- architecture rtl

  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  correlator_EL_inst : correlator_EL
    port map (
      rst               => rst,
      clk               => clk,
      enable            => enable,
      signal_IN         => signal_IN,
      delay_CA          => delay_CA,
      update_up         => update_up_i,
      update_down       => update_down_i,
      ready_EL          => ready_EL_i,
      correlation_early => correlation_early_i,
      correlation_late  => correlation_late_i,
      delay             => delay);

  logic_control_inst : logic_control
    port map (
      rst               => rst,
      clk               => clk,
      enable            => ready_EL_i,
      correlation_early => correlation_early_i,
      correlation_late  => correlation_late_i,
      update_up         => update_up_i,
      update_down       => update_down_i);

end architecture rtl;
```

Código fuente A.11: correlator_EL.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- CORRELATOR EARLY - LATE
------------------------------------------------------------------------------
entity correlator_EL is

  port (
    rst               : in  std_logic;  -- rst signal
    clk               : in  std_logic;  -- clk signal
    enable            : in  std_logic;  -- enable tracking flag
    signal_IN         : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
    delay_CA          : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay estimation
    update_up         : in  std_logic;  -- increase delay estimation
```

```
    update_down       : in  std_logic;  -- decrease delay estimation
    ready_EL          : out std_logic;  -- early-late correlation values ready
    correlation_early : out std_logic_vector(bits_corr-1 downto 0);     -- early correlation value
    correlation_late  : out std_logic_vector(bits_corr-1 downto 0);     -- late correlation value
    delay             : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation

end correlator_EL;


architecture rtl of correlator_EL is

  ------------------------------------------------------------------------------
  -- Component Declaration
  ------------------------------------------------------------------------------
  component correlator_early
    port (
      rst         : in  std_logic;      -- rst signal
      clk         : in  std_logic;      -- clk signal
      enable      : in  std_logic;      -- enable correlation flag
      signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
      delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
      ready       : out std_logic;                                 -- early correlation value ready
      correlation : out std_logic_vector(bits_corr-1 downto 0));   -- early correlation value
  end component;

  component delay_estimation
    port (
      rst         : in  std_logic;      -- rst signal
      clk         : in  std_logic;      -- clk signal
      enable      : in  std_logic;      -- enable correlation flag
      signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
      delay_CA    : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay estimation
      update_up   : in  std_logic;                                   -- increase delay estimation
      update_down : in  std_logic;                                   -- decrease delay estimation
      delay       : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation
  end component;

  component correlator_late
    port (
      rst         : in  std_logic;      -- rst signal
      clk         : in  std_logic;      -- clk signal
      enable      : in  std_logic;      -- enable correlation signal
      signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
      delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
      ready       : out std_logic;                                 -- late correlation value ready
      correlation : out std_logic_vector(bits_corr-1 downto 0));   -- late correlation value
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal signal_IN_1     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_2     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_3     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_4     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_track : std_logic_vector(bits_adc-1 downto 0);
  signal ready_E         : std_logic;
  signal ready_L         : std_logic;
  signal delay_i         : std_logic_vector(bits_samples-1 downto 0);


begin  -- rtl

  ------------------------------------------------------------------------------
  -- Signals Instantiation
  ------------------------------------------------------------------------------
  signal_IN_track <= signal_IN_4;
  ready_EL        <= ready_E and ready_L;
  delay           <= delay_i;
```

```
  ------------------------------------------------------------------------------
  -- Component Instantiation
  ------------------------------------------------------------------------------
  correlator_early_inst : correlator_early
    port map (
      rst         => rst,
      clk         => clk,
      enable      => enable,
      signal_IN   => signal_IN_track,
      delay       => delay_i,
      ready       => ready_E,
      correlation => correlation_early);

  delay_estimation_inst : delay_estimation
    port map (
      rst         => rst,
      clk         => clk,
      enable      => enable,
      signal_IN   => signal_IN_track,
      delay_CA    => delay_CA,
      update_up   => update_up,
      update_down => update_down,
      delay       => delay_i);

  correlator_late_inst : correlator_late
    port map (
      rst         => rst,
      clk         => clk,
      enable      => enable,
      signal_IN   => signal_IN_track,
      delay       => delay_i,
      ready       => ready_L,
      correlation => correlation_late);


  ------------------------------------------------------------------------------
  -- Delay Process To Synchronize Input & Memory Signals (4 MEM Latency Cycles)
  -- 2 cycles from C/A Synchronization Stage and 2 more cycles from this Stage
  ------------------------------------------------------------------------------
  delaying : process (clk, rst) is
  begin  -- process delaying

    if rst = '1' then                   -- asynchronous reset (active high)

      signal_IN_1 <= (others => '0');
      signal_IN_2 <= (others => '0');
      signal_IN_3 <= (others => '0');
      signal_IN_4 <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      signal_IN_1 <= signal_IN;         -- Latency from C/A Synchronization Stage
      signal_IN_2 <= signal_IN_1;
      signal_IN_3 <= signal_IN_2;       -- Latency from Tracking ELDLL Stage
      signal_IN_4 <= signal_IN_3;

    end if;

  end process delaying;

end rtl;
```

Código fuente A.12: correlator_early.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

library UNISIM;
```

```
use UNISIM.vcomponents.all;

library UNIMACRO;
use UNIMACRO.vcomponents.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- CORRELATOR EARLY
--------------------------------------------------------------------------------
entity correlator_early is

  port (
    rst         : in  std_logic;        -- rst signal
    clk         : in  std_logic;        -- clk signal
    enable      : in  std_logic;        -- enable correlation flag
    signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
    delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
    ready       : out std_logic;                                 -- early correlation value ready
    correlation : out std_logic_vector(bits_corr-1 downto 0));   -- early correlation value

end correlator_early;


architecture rtl of correlator_early is

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  component buffer_MEM_EL
    port (
      rst    : in  std_logic;           -- rst signal
      clk    : in  std_logic;           -- clk signal
      enable : in  std_logic;           -- enable signal
      delay  : in  std_logic_vector(bits_samples-1 downto 0);  -- continuous delay estimation
      sample : out std_logic_vector(bits_adc-1 downto 0));     -- output sample
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal signal_MEM  : std_logic_vector(bits_adc-1 downto 0);
  signal result      : std_logic_vector(bits_corr-1 downto 0);
  signal delay_early : std_logic_vector(bits_samples-1 downto 0);
  signal counter     : std_logic_vector(bits_samples downto 0);
  signal sclr_macc   : std_logic;


begin  -- rtl

  ------------------------------------------------------------------------------
  -- Signals Instantiation
  ------------------------------------------------------------------------------
  delay_early <= delay - 1;


  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  buffer_MEM_EL_inst : buffer_MEM_EL
    port map (
      rst    => rst,
      clk    => clk,
      enable => enable,
      delay  => delay_early,
      sample => signal_MEM);
```

```
-- MACC_MACRO: Multiple Accumulate Function implemented in a DSP48E (Spartan 6)
MACC_MACRO_inst : MACC_MACRO
  generic map (
    DEVICE  => "SPARTAN6",            -- Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6"
    LATENCY => 1,                     -- Desired clock cycle latency, 1-4
    WIDTH_A => bits_adc,              -- Multiplier A-input bus width, 1-25
    WIDTH_B => bits_adc,              -- Multiplier B-input bus width, 1-18
    WIDTH_P => bits_corr)             -- Accumulator output bus width, 1-48
  port map (
    P         => result,              -- MACC ouput bus, width determined by WIDTH_P generic
    A         => signal_IN,           -- MACC input A bus, width determined by WIDTH_A generic
    ADDSUB    => '1',                 -- 1-bit add/sub input, high selects add, low selects subtract
    B         => signal_MEM,          -- MACC input B bus, width determined by WIDTH_B generic
    CARRYIN   => '0',                 -- 1-bit carry-in input to accumulator
    CE        => enable,              -- 1-bit active high input clock enable
    CLK       => clk,                 -- 1-bit positive edge clock input
    LOAD      => '0',                 -- 1-bit active high input load accumulator enable
    LOAD_DATA => (others => '0'),     -- Load accumulator input data, width determined by WIDTH_P generic
    RST       => sclr_macc);          -- 1-bit input active high reset
-- End of MACC_MACRO_inst instantiation


--------------------------------------------------------------------------------
-- End of Samples Processing Process
--------------------------------------------------------------------------------
processing_end : process (clk, rst)
begin  -- process processing_end

  if rst = '1' then                   -- asynchronous reset (active high)

    counter     <= (others => '0');
    correlation <= (others => '0');
    ready       <= '0';

  elsif clk'event and clk = '1' then  -- rising clock edge

    if enable = '1' then

      counter <= counter + 1;

      if counter = n_samples-1 then

        counter <= (others => '0');

      end if;

    end if;

    if counter = n_samples-1 then

      correlation <= result;
      ready       <= '1';

    else

      ready <= '0';

    end if;

  end if;

end process processing_end;


--------------------------------------------------------------------------------
-- MACC Reset Process
--------------------------------------------------------------------------------
macc_reset : process (clk, rst)
begin  -- process macc_reset
```

```
    if rst = '1' then                   -- asynchronous reset (active high)
      sclr_macc <= '1';
    elsif clk'event and clk = '1' then  -- rising clock edge
      if counter = n_samples-1 then
        sclr_macc <= '1';
      else
        sclr_macc <= '0';
      end if;
    end if;
  end process macc_reset;
end rtl;
```

Código fuente A.13: delay_estimation.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- DELAY ESTIMATION
-------------------------------------------------------------------------------
entity delay_estimation is

  port (
    rst         : in  std_logic;        -- rst signal
    clk         : in  std_logic;        -- clk signal
    enable      : in  std_logic;        -- enable correlation signal
    signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);      -- input signal
    delay_CA    : in  std_logic_vector(bits_samples-1 downto 0);  -- C/A delay estimation
    update_up   : in  std_logic;        -- increase delay estimation
    update_down : in  std_logic;        -- decrease delay estimation
    delay       : out std_logic_vector(bits_samples-1 downto 0)); -- continuous delay estimation

end delay_estimation;


architecture rtl of delay_estimation is

  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal delay_actual : std_logic_vector(bits_samples downto 0);
  signal update_delay : std_logic_vector(bits_samples downto 0);
  signal counter      : std_logic_vector(bits_samples downto 0);


begin  -- rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
  delay_actual <= ('0' & delay_CA) + update_delay;
  delay        <= delay_actual(bits_samples-1 downto 0);


  -----------------------------------------------------------------------------
  -- Update Delay Estimation Process
```

```
  ---------------------------------------------------------------------------
  update : process (clk, rst)
  begin  -- process update

    if rst = '1' then                   -- asynchronous reset (active high)

      update_delay <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if counter = 1 then

        if update_up = '1' then

          update_delay <= update_delay + 1;

        end if;

        if update_down = '1' then

          update_delay <= update_delay - 1;

        end if;

      end if;

    end if;

  end process update;


  process (clk, rst)
  begin  -- process

    if rst = '1' then                   -- asynchronous reset (active high)

      counter <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        if counter < n_samples-1 then

          counter <= counter + 1;

        else

          counter <= (others => '0');

        end if;

      end if;

    end if;

  end process;

end rtl;
```

Código fuente A.14: correlator_late.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

library UNISIM;
use UNISIM.vcomponents.all;
```

```
library UNIMACRO;
use UNIMACRO.vcomponents.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- CORRELATOR LATE
--------------------------------------------------------------------------------
entity correlator_late is

  port (
    rst         : in  std_logic;        -- rst signal
    clk         : in  std_logic;        -- clk signal
    enable      : in  std_logic;        -- enable correlation signal
    signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
    delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
    ready       : out std_logic;                                 -- late correlation value ready
    correlation : out std_logic_vector(bits_corr-1 downto 0));   -- late correlation value

end correlator_late;


architecture rtl of correlator_late is

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  component buffer_MEM_EL
    port (
      rst    : in  std_logic;           -- rst signal
      clk    : in  std_logic;           -- clk signal
      enable : in  std_logic;           -- enable signal
      delay  : in  std_logic_vector(bits_samples-1 downto 0);  -- continuous delay estimation
      sample : out std_logic_vector(bits_adc-1 downto 0));     -- output sample
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal signal_MEM : std_logic_vector(bits_adc-1 downto 0);
  signal result     : std_logic_vector(bits_corr-1 downto 0);
  signal delay_late : std_logic_vector(bits_samples-1 downto 0);
  signal counter    : std_logic_vector(bits_samples downto 0);
  signal sclr_macc  : std_logic;


begin  -- rtl

  ------------------------------------------------------------------------------
  -- Signals Instantiation
  ------------------------------------------------------------------------------
  delay_late <= delay + 1;


  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  buffe_MEM_EL_inst : buffer_MEM_EL
    port map (
      rst    => rst,
      clk    => clk,
      enable => enable,
      delay  => delay_late,
      sample => signal_MEM);

  -- MACC_MACRO: Multiple Accumulate Function implemented in a DSP48E (Spartan 6)
  MACC_MACRO_inst : MACC_MACRO
    generic map (
      DEVICE  => "SPARTAN6",            -- Target Device: "VIRTEX5", "VIRTEX6", "SPARTAN6"
```

```
    LATENCY => 1,                    -- Desired clock cycle latency, 1-4
    WIDTH_A => bits_adc,             -- Multiplier A-input bus width, 1-25
    WIDTH_B => bits_adc,             -- Multiplier B-input bus width, 1-18
    WIDTH_P => bits_corr)            -- Accumulator output bus width, 1-48
  port map (
    P         => result,             -- MACC ouput bus, width determined by WIDTH_P generic
    A         => signal_IN,          -- MACC input A bus, width determined by WIDTH_A generic
    ADDSUB    => '1',                -- 1-bit add/sub input, high selects add, low selects subtract
    B         => signal_MEM,         -- MACC input B bus, width determined by WIDTH_B generic
    CARRYIN   => '0',                -- 1-bit carry-in input to accumulator
    CE        => enable,             -- 1-bit active high input clock enable
    CLK       => clk,                -- 1-bit positive edge clock input
    LOAD      => '0',                -- 1-bit active high input load accumulator enable
    LOAD_DATA => (others => '0'),    -- Load accumulator input data, width determined by WIDTH_P generic
    RST       => sclr_macc);         -- 1-bit input active high reset
-- End of MACC_MACRO_inst instantiation


--------------------------------------------------------------------------------
-- End of Samples Processing Process
--------------------------------------------------------------------------------
processing_end : process (clk, rst)
begin  -- process processing_end

  if rst = '1' then                  -- asynchronous reset (active high)

    counter     <= (others => '0');
    correlation <= (others => '0');
    ready       <= '0';

  elsif clk'event and clk = '1' then  -- rising clock edge

    if enable = '1' then

      counter <= counter + 1;

      if counter = n_samples-1 then

        counter <= (others => '0');

      end if;

    end if;

    if counter = n_samples-1 then

      correlation <= result;
      ready       <= '1';

    else

      ready <= '0';

    end if;

  end if;

end process processing_end;


--------------------------------------------------------------------------------
-- MACC Reset Process
--------------------------------------------------------------------------------
macc_reset : process (clk, rst)
begin  -- process macc_reset

  if rst = '1' then                  -- asynchronous reset (active high)

    sclr_macc <= '1';
```

```
    elsif clk'event and clk = '1' then  -- rising clock edge

      if counter = n_samples-1 then

        sclr_macc <= '1';

      else

        sclr_macc <= '0';

      end if;

    end if;

  end process macc_reset;

end rtl;
```

Código fuente A.15: buffer_MEM_EL.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- BUFFER CORRELATION MEM EL
-------------------------------------------------------------------------------
entity buffer_MEM_EL is

  port (
    rst    : in  std_logic;             -- rst signal
    clk    : in  std_logic;             -- clk signal
    enable : in  std_logic;             -- enable signal
    delay  : in  std_logic_vector(bits_samples-1 downto 0);  -- continuous delay estimation
    sample : out std_logic_vector(bits_adc-1 downto 0));     -- output sample

end buffer_MEM_EL;


architecture rtl of buffer_MEM_EL is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component memory_local
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples downto 0);
      data_out : out std_logic_vector(bits_adc-1 downto 0));
  end component;


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal address : std_logic_vector(bits_samples downto 0);
  signal counter : std_logic_vector(bits_samples downto 0);


begin  -- rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
```

```
  address <= (counter + delay) when ((counter + delay) < n_samples) else ((counter + delay) - n_samples);


  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  memory_local_inst : memory_local
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      r_w      => '0',
      address  => address,
      data_out => sample);


  ------------------------------------------------------------------------------
  -- Local Signal Reading Process
  ------------------------------------------------------------------------------
  process (clk, rst)
  begin  -- process

    if rst = '1' then                   -- asynchronous reset (active high)

      counter <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        if counter < n_samples-1 then

          counter <= counter + 1;

        else

          counter <= (others => '0');

        end if;

      end if;

    end if;

  end process;

end rtl;
```

Código fuente A.16: logic_control.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- LOGIC CONTROL
------------------------------------------------------------------------------
entity logic_control is

  port (
    rst              : in  std_logic;  -- reset signal
    clk              : in  std_logic;  -- clock signal
    enable           : in  std_logic;  -- E-L correlation ready
    correlation_early : in  std_logic_vector(bits_corr-1 downto 0);  -- correlation early value
    correlation_late  : in  std_logic_vector(bits_corr-1 downto 0);  -- correlation late value
    update_up        : out std_logic;  -- increase delay estimation
    update_down      : out std_logic); -- decrease delay estimation
```

```
end logic_control;


architecture rtl of logic_control is

  ---------------------------------------------------------------------------
  -- Component Declaration
  ---------------------------------------------------------------------------
  component filter_loop is
    port (
      rst      : in  std_logic;                                -- reset signal
      clk      : in  std_logic;                                -- clock signal
      enable   : in  std_logic;                                -- enable signal
      data_IN  : in  std_logic_vector(bits_corr-1 downto 0);   -- input data
      data_OUT : out std_logic_vector(bits_corr-1 downto 0));  -- output data
  end component;


  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal difference_EL   : std_logic_vector(bits_corr-1 downto 0);
  signal difference_EL_i : std_logic_vector(bits_corr-1 downto 0);
  signal discriminator   : std_logic_vector(bits_corr-1 downto 0);
  signal discriminator_i : std_logic_vector(bits_corr-1 downto 0);
  signal sum_EL          : std_logic_vector(bits_corr-1 downto 0);
  signal update          : std_logic;


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  filter_loop_inst : filter_loop
    port map (
      rst      => rst,
      clk      => clk,
      enable   => '1',
      data_IN  => difference_EL_i,
      data_OUT => discriminator);


  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  difference_EL <= correlation_early - correlation_late;
  sum_EL        <= correlation_early + correlation_late;


  ---------------------------------------------------------------------------
  -- Update Control Process
  ---------------------------------------------------------------------------
  control : process (clk, rst)
  begin  -- process control

    if rst = '1' then                   -- asynchronous reset (active high)

      update      <= '0';
      update_up   <= '0';
      update_down <= '0';

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        if discriminator_i > threshold_EL and update = '0' then

          update_up   <= '0';
```

```
          update_down <= '1';
          update      <= '1';

        elsif discriminator_i < -threshold_EL and update = '0' then

          update_up   <= '1';
          update_down <= '0';
          update      <= '1';

        elsif discriminator_i < threshold_EL and discriminator_i > -threshold_EL then

          update_up   <= '0';
          update_down <= '0';
          update      <= '0';

        end if;

      else

        update_up   <= '0';
        update_down <= '0';

      end if;

    end if;

  end process control;


  process (clk, rst) is
  begin  -- process

    if rst = '1' then                   -- asynchronous reset (active high)

      difference_EL_i <= (others => '0');
      discriminator_i <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      difference_EL_i <= difference_EL;
      discriminator_i <= discriminator;

    end if;

  end process;

end rtl;
```

Código fuente A.17: filter_loop.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- LOOP FILTER
-------------------------------------------------------------------------------
entity filter_loop is

  port (
    rst      : in  std_logic;                                -- reset signal
    clk      : in  std_logic;                                -- clock signal
    enable   : in  std_logic;                                -- enable signal
    data_IN  : in  std_logic_vector(bits_corr-1 downto 0);   -- input data
    data_OUT : out std_logic_vector(bits_corr-1 downto 0));  -- output data

end entity filter_loop;
```

```
architecture rtl of filter_loop is

  ---------------------------------------------------------------------------
  -- Constants Declaration
  ---------------------------------------------------------------------------
  constant num_LOOP_1 : std_logic_vector(bits_filter-1 downto 0) := X"0066";
  constant num_LOOP_2 : std_logic_vector(bits_filter-1 downto 0) := X"0066";
  constant den_LOOP_1 : std_logic_vector(bits_filter-1 downto 0) := X"7FFF";
  constant den_LOOP_2 : std_logic_vector(bits_filter-1 downto 0) := X"80cd";


  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal x_n_i : std_logic_vector(2*bits_adc-1 downto 0);              -- input signal
  signal x_n   : std_logic_vector(2*bits_adc+bits_filter-1 downto 0);  -- input signal
  signal x_n_1 : std_logic_vector(2*bits_adc+bits_filter-1 downto 0);  -- input signal delayed 1 sample
  signal y_n   : std_logic_vector(2*bits_adc+bits_filter-1 downto 0);  -- output signal
  signal y_n_1 : std_logic_vector(2*bits_adc+bits_filter-1 downto 0);  -- output signal delayed 1 sample


begin  -- architecture rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  data_OUT <= y_n(2*bits_adc+bits_filter-2 downto bits_filter-1) & X"00";


  ---------------------------------------------------------------------------
  -- IIR Loop Filter Process
  ---------------------------------------------------------------------------
  iir_loop_filter : process (clk, rst) is
  begin  -- process iir_loop_filter

    if rst = '1' then                   -- asynchronous reset (active high)

      x_n   <= (others => '0');
      x_n_i <= (others => '0');
      x_n_1 <= (others => '0');
      y_n   <= (others => '0');
      y_n_1 <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        x_n_i <= data_IN(bits_corr-1 downto bits_samples);
        x_n   <= num_LOOP_1*data_IN(bits_corr-1 downto bits_samples);
        x_n_1 <= num_LOOP_2*x_n_i;
        y_n   <= x_n + x_n_1 - y_n_1;
        y_n_1 <= den_LOOP_2*y_n(2*bits_adc+bits_filter-2 downto bits_filter-1);

      end if;

    end if;

  end process iir_loop_filter;

end architecture rtl;
```

Código fuente A.18: demodulation_DSSS.vhd

```
library ieee;
use ieee.std_logic_1164.all;

library UNISIM;
```

```
use UNISIM.vcomponents.all;

library XilinxCoreLib;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- DEMODULATION DSSS STAGE
-------------------------------------------------------------------------------
entity demodulation_DSSS is

  port (
    rst         : in  std_logic;        -- reset signal
    clk         : in  std_logic;        -- clock signal
    enable      : in  std_logic;        -- enable DSSS demodulation stage
    signal_IN   : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
    delay       : in  std_logic_vector(bits_samples-1 downto 0); -- continuous delay estimation
    ready_DSSS  : out std_logic;                                 -- DSSS demodulation ready
    signal_DSSS : out std_logic_vector(bits_adc-1 downto 0));    -- DSSS demodulated input signal

end demodulation_DSSS;


architecture rtl of demodulation_DSSS is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component buffer_MEM_PRN
    port (
      rst    : in  std_logic;           -- rst signal
      clk    : in  std_logic;           -- clk signal
      enable : in  std_logic;           -- enable signal
      delay  : in  std_logic_vector(bits_samples-1 downto 0);  -- continuous delay estimation
      sample : out std_logic_vector(bits_adc-1 downto 0));     -- output sample
  end component;

  component multiplier
    port (
      clk  : in  std_logic;
      a    : in  std_logic_vector(7 downto 0);
      b    : in  std_logic_vector(7 downto 0);
      ce   : in  std_logic;
      sclr : in  std_logic;
      p    : out std_logic_vector(15 downto 0));
  end component;


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal signal_IN_1     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_2     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_3     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_4     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_track : std_logic_vector(bits_adc-1 downto 0);
  signal code_PRN        : std_logic_vector(bits_adc-1 downto 0);
  signal code_PRN_i      : std_logic_vector(bits_adc-1 downto 0);
  signal result          : std_logic_vector(2*bits_adc-1 downto 0);
  signal ready_DSSS_i    : std_logic;


begin  -- rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
  signal_DSSS <= result(bits_adc-1 downto 0);
  ready_DSSS  <= ready_DSSS_i;
```

```
-------------------------------------------------------------------------------
-- Components Instantiation
-------------------------------------------------------------------------------
buffer_MEM_PRN_inst : buffer_MEM_PRN
  port map (
    rst    => rst,
    clk    => clk,
    enable => enable,
    delay  => delay,
    sample => code_PRN_i);

multiplier_inst : multiplier
  port map (
    clk  => clk,
    a    => signal_IN_track,
    b    => code_PRN,
    ce   => enable,
    sclr => rst,
    p    => result);


-------------------------------------------------------------------------------
-- Delay Process To Synchronize Input & Memory Signals (4 MEM Latency Cycles)
-- 2 cycles from C/A Synchronization Stage and 2 more cycles from this Stage
-------------------------------------------------------------------------------
delaying : process (clk, rst) is
begin  -- process delay

  if rst = '1' then                   -- asynchronous reset (active high)

    signal_IN_1     <= (others => '0');
    signal_IN_2     <= (others => '0');
    signal_IN_3     <= (others => '0');
    signal_IN_4     <= (others => '0');
    signal_IN_track <= (others => '0');
    code_PRN        <= (others => '0');

  elsif clk'event and clk = '1' then  -- rising clock edge

    signal_IN_1 <= signal_IN;         -- Latency from C/A Synchronization Stage
    signal_IN_2 <= signal_IN_1;
    signal_IN_3 <= signal_IN_2;       -- Latency from Demodulation DSSS Stage
    signal_IN_4 <= signal_IN_3;

    -- Input Signal Correction When IN = -128 & PRN = -1
    -- To Avoid Overflow (Adds An Extra Latency Cycle)
    if signal_IN_4 = X"80" and code_PRN_i = X"FF" then

      signal_IN_track <= X"81";

    else

      signal_IN_track <= signal_IN_4;

    end if;

    code_PRN <= code_PRN_i;

  end if;

end process delaying;


-------------------------------------------------------------------------------
-- DSSS Demodulation Ready Process To Indicate When It Begins
-------------------------------------------------------------------------------
dsss : process (clk, rst)
begin  -- process dsss
```

```
    if rst = '1' then                   -- asynchronous reset (active high)
      ready_DSSS_i <= '0';
    elsif clk'event and clk = '1' then  -- rising clock edge
      if ready_DSSS_i = '0' and code_PRN /= X"00" then
        ready_DSSS_i <= '1';
      end if;
    end if;
  end process dsss;
end rtl;
```

Código fuente A.19: buffer_MEM_PRN.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- BUFFER CORRELATION MEM EL
-------------------------------------------------------------------------------
entity buffer_MEM_PRN is

  port (
    rst    : in  std_logic;             -- rst signal
    clk    : in  std_logic;             -- clk signal
    enable : in  std_logic;             -- enable signal
    delay  : in  std_logic_vector(bits_samples-1 downto 0);  -- continuous delay estimation
    sample : out std_logic_vector(bits_adc-1 downto 0));     -- output sample

end buffer_MEM_PRN;


architecture rtl of buffer_MEM_PRN is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component memory_PRN
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples downto 0);
      data_out : out std_logic_vector(bits_adc-1 downto 0));
  end component;


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal address : std_logic_vector(bits_samples downto 0);
  signal counter : std_logic_vector(bits_samples downto 0);


begin  -- rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
```

```
  address <= (counter + delay) when ((counter + delay) < n_samples) else ((counter + delay) - n_samples);


  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  memory_PRN_inst : memory_PRN
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      r_w      => '0',
      address  => address,
      data_out => sample);


  ------------------------------------------------------------------------------
  -- PRN Code Reading Process
  ------------------------------------------------------------------------------
  mem_prn : process (clk, rst)
  begin  -- process mem_prn

    if rst = '1' then                   -- asynchronous reset (active high)

      counter <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        if counter < n_samples-1 then

          counter <= counter + 1;

        else

          counter <= (others => '0');

        end if;

      end if;

    end if;

  end process mem_prn;

end rtl;
```

Código fuente A.20: memory_PRN.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- PRN MEMORY (BRAM)
------------------------------------------------------------------------------
entity memory_PRN is

  port (
    rst      : in  std_logic;
    clk      : in  std_logic;
    enable   : in  std_logic;
    r_w      : in  std_logic;
```

```
    address  : in  std_logic_vector(bits_samples downto 0);
    data_out : out std_logic_vector(bits_adc-1 downto 0));

end memory_PRN;


architecture rtl of memory_PRN is

  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal r_w_i      : std_logic_vector(3 downto 0);
  signal address_i  : std_logic_vector(13 downto 0);
  signal data_in_i  : std_logic_vector(31 downto 0);
  signal data_out_i : std_logic_vector(31 downto 0);


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  r_w_i     <= (others => r_w);
  address_i <= "00" & address & "000";
  data_out  <= data_out_i(bits_adc-1 downto 0);


  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  -- RAMB16BWER: 16K-bit Data and 2K-bit Parity Configurable Synchronous Dual Port
  -- Block RAM with Optional Output Registers
  -- Spartan-6, Xilinx HDL Libraries Guide, version 12.1
  mem_local_inst : RAMB16BWER
    generic map (
      -- DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
      DATA_WIDTH_A        => 9,
      DATA_WIDTH_B        => 0,
      -- DOA_REG/DOB_REG: Optional output register (0 or 1)
      DOA_REG             => 1,
      DOB_REG             => 0,
      -- EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
      EN_RSTRAM_A         => true,
      EN_RSTRAM_B         => false,
      -- INIT_00 to INIT_3F: Initial memory contents: fm = 25 MHz, fs = 100 MHz, fchip = fm
      INIT_00             => X"ffffffffffffffff01010101ffffffffffffffffffffffffffffffffffffffff",
      INIT_01             => X"01010101ffffffffffffffffffffffff0101010101010101ffffffffffffffff",
      INIT_02             => X"0101010101010101010101010101010101ffffffffffffff01010101ffffffff",
      INIT_03             => X"0101010101010101010101010101ffffffffffffffffffffffff01010101ffffffff",
      INIT_04             => X"01010101ffffffff01010101ffffffffffffffff01010101ffffffffffffffff",
      INIT_05             => X"0101010101010101ffffffff0101010101010101010101ffffffff01010101",
      INIT_06             => X"01010101ffffffff01010101ffffffff0101010101010101ffffffffffffffff",
      INIT_07             => X"0000000001010101010101010101010101010101010101010101010101ffffffff",
      -- INIT_A/INIT_B: Initial values on output port
      INIT_A              => X"000000000",
      INIT_B              => X"000000000",
      -- INIT_FILE: Optional file used to specify initial RAM contents
      INIT_FILE           => "NONE",
      -- RSTTYPE: "SYNC" or "ASYNC"
      RSTTYPE             => "SYNC",
      -- RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
      RST_PRIORITY_A      => "SR",
      RST_PRIORITY_B      => "CE",
      -- SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY", "GENERATE_X_ONLY" or "NONE"
      SIM_COLLISION_CHECK => "ALL",
      -- SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
      SIM_DEVICE          => "SPARTAN6",
      -- SRVAL_A/SRVAL_B: Set/Reset value for RAM output
      SRVAL_A             => X"000000000",
      SRVAL_B             => X"000000000",
```

```
      -- WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
      WRITE_MODE_A        => "NO_CHANGE",
      WRITE_MODE_B        => "NO_CHANGE")

    port map (
      DOA    => data_out_i,             -- 32-bit A port data output
      DOB    => open,                   -- 32-bit B port data output
      DOPA   => open,                   -- 4-bit A port parity output
      DOPB   => open,                   -- 4-bit B port parity output
      ADDRA  => address_i,              -- 14-bit A port address input
      ADDRB  => "00" & x"000",          -- 14-bit B port address input
      CLKA   => clk,                    -- 1-bit A port clock input
      CLKB   => '0',                    -- 1-bit B port clock input
      DIA    => x"00000000",            -- 32-bit A port data input
      DIB    => x"00000000",            -- 32-bit B port data input
      DIPA   => "0000",                 -- 4-bit A port parity input
      DIPB   => "0000",                 -- 4-bit B port parity input
      ENA    => enable,                 -- 1-bit A port enable input
      ENB    => '0',                    -- 1-bit B port enable input
      REGCEA => '1',                    -- 1-bit A port register clock enable input
      REGCEB => '0',                    -- 1-bit B port register clock enable input
      RSTA   => rst,                    -- 1-bit A port register set/reset input
      RSTB   => '0',                    -- 1-bit B port register set/reset input
      WEA    => r_w_i,                  -- 4-bit Port A byte-wide write enable input
      WEB    => r_w_i);                 -- 4-bit Port B byte-wide write enable input
  -- End of RAMB16BWER_inst instantiation

end rtl;
```

Código fuente A.21: multiplier.vhd

```
library ieee;
use ieee.std_logic_1164.all;
-- synthesis translate_off
library XilinxCoreLib;
-- synthesis translate_on

--------------------------------------------------------------------------------
-- MULTIPLIER
--------------------------------------------------------------------------------
entity multiplier is

  port (
    clk  : in  std_logic;
    a    : in  std_logic_vector(7 downto 0);
    b    : in  std_logic_vector(7 downto 0);
    ce   : in  std_logic;
    sclr : in  std_logic;
    p    : out std_logic_vector(15 downto 0));

end multiplier;


architecture multiplier_a of multiplier is

  --------------------------------------------------------------------------------
  -- Components Declaration
  --------------------------------------------------------------------------------
  -- synthesis translate_off
  component wrapped_multiplier
    port (
      clk  : in  std_logic;
      a    : in  std_logic_vector(7 downto 0);
      b    : in  std_logic_vector(7 downto 0);
      ce   : in  std_logic;
      sclr : in  std_logic;
      p    : out std_logic_vector(15 downto 0));
  end component;
```

```
  -- Configuration specification
  for all : wrapped_multiplier use entity XilinxCoreLib.mult_gen_v11_2(behavioral)
    generic map(
      c_a_width           => 8,
      c_b_type            => 0,
      c_ce_overrides_sclr => 0,
      c_has_sclr          => 1,
      c_round_pt          => 0,
      c_model_type        => 0,
      c_out_high          => 15,
      c_verbosity         => 0,
      c_mult_type         => 1,
      c_ccm_imp           => 0,
      c_latency           => 1,
      c_has_ce            => 1,
      c_has_zero_detect   => 0,
      c_round_output      => 0,
      c_optimize_goal     => 1,
      c_xdevicefamily     => "spartan6",
      c_a_type            => 0,
      c_out_low           => 0,
      c_b_width           => 8,
      c_b_value           => "10000001");
  -- synthesis translate_on


begin

  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  -- synthesis translate_off
  U0 : wrapped_multiplier
    port map (
      clk  => clk,
      a    => a,
      b    => b,
      ce   => ce,
      sclr => sclr,
      p    => p);
  -- synthesis translate_on

end multiplier_a;
```

Código fuente A.22: phase_estimation.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- PHASE ESTIMATION STAGE
------------------------------------------------------------------------------
entity phase_estimation is

  port (
    rst       : in  std_logic;          -- reset signal
    clk       : in  std_logic;          -- clock signal
    enable    : in  std_logic;          -- enable phase estimation stage
    signal_IN : in  std_logic_vector(bits_adc-1 downto 0);     -- input signal
    phase     : out std_logic_vector(bits_phase-1 downto 0));  -- phase estimated

end entity phase_estimation;
```

```
architecture rtl of phase_estimation is

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  component demodulation_IQ is
    port (
      rst         : in  std_logic;      -- reset signal
      clk         : in  std_logic;      -- clock signal
      enable      : in  std_logic;      -- enable I/Q demodulation flag
      signal_IN   : in  std_logic_vector(bits_adc-1 downto 0); -- DSSS demodulated input signal
      signal_IN_I : out std_logic_vector(bits_iq-1 downto 0);  -- I demodulated input signal
      signal_IN_Q : out std_logic_vector(bits_iq-1 downto 0)); -- Q demodulated input signal
  end component;

  component filter_IQ is
    port (
      rst        : in  std_logic;                              -- reset signal
      clk        : in  std_logic;                              -- clock signal
      enable     : in  std_logic;                              -- enable flag
      data_IN_I  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data I
      data_IN_Q  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data Q
      data_OUT_I : out std_logic_vector(bits_iq-1 downto 0);   -- output data I
      data_OUT_Q : out std_logic_vector(bits_iq-1 downto 0));  -- output data Q
  end component;

  component phase_extractor is
    port (
      rst         : in  std_logic;      -- reset signal
      clk         : in  std_logic;      -- clock signal
      enable      : in  std_logic;      -- enable arctan
      signal_IN_I : in  std_logic_vector(bits_iq-1 downto 0);     -- input signal demodulated I
      signal_IN_Q : in  std_logic_vector(bits_iq-1 downto 0);     -- input signal demodulated Q
      phase       : out std_logic_vector(bits_phase-1 downto 0)); -- phase estimated
  end component;


  ------------------------------------------------------------------------------
  -- Signals Declaration
  ------------------------------------------------------------------------------
  signal signal_IN_I_i : std_logic_vector(bits_iq-1 downto 0);
  signal signal_IN_Q_i : std_logic_vector(bits_iq-1 downto 0);
  signal signal_IN_I_f : std_logic_vector(bits_iq-1 downto 0);
  signal signal_IN_Q_f : std_logic_vector(bits_iq-1 downto 0);


begin  -- architecture rtl

  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  demodulation_IQ_inst : demodulation_IQ
    port map (
      rst         => rst,
      clk         => clk,
      enable      => enable,
      signal_IN   => signal_IN,
      signal_IN_I => signal_IN_I_i,
      signal_IN_Q => signal_IN_Q_i);

  filter_IQ_inst : filter_IQ
    port map (
      rst        => rst,
      clk        => clk,
      enable     => enable,
      data_IN_I  => signal_IN_I_i,
      data_IN_Q  => signal_IN_Q_i,
      data_OUT_I => signal_IN_I_f,
      data_OUT_Q => signal_IN_Q_f);
```

```
  phase_extractor_inst : phase_extractor
    port map (
      rst         => rst,
      clk         => clk,
      enable      => enable,
      signal_IN_I => signal_IN_I_f,
      signal_IN_Q => signal_IN_Q_f,
      phase       => phase);

end architecture rtl;
```

Código fuente A.23: demodulation_IQ.vhd

```
library ieee;
use ieee.std_logic_1164.all;

library UNISIM;
use UNISIM.vcomponents.all;

library XilinxCoreLib;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- DEMODULATION IQ
-------------------------------------------------------------------------------
entity demodulation_IQ is

  port (
    rst         : in  std_logic;        -- reset signal
    clk         : in  std_logic;        -- clock signal
    enable      : in  std_logic;        -- enable I/Q demodulation signal
    signal_IN   : in  std_logic_vector(bits_adc-1 downto 0); -- DSSS demodulated input signal
    signal_IN_I : out std_logic_vector(bits_iq-1 downto 0);  -- I demodulated input signal
    signal_IN_Q : out std_logic_vector(bits_iq-1 downto 0)); -- Q demodulated input signal

end demodulation_IQ;


architecture rtl of demodulation_IQ is

  -------------------------------------------------------------------------------
  -- Components Declaration
  -------------------------------------------------------------------------------
  component buffer_MEM_IQ
    port (
      rst      : in  std_logic;         -- rst signal
      clk      : in  std_logic;         -- clk signal
      enable   : in  std_logic;         -- enable signal
      sample_I : out std_logic_vector(bits_adc-1 downto 0);   -- output sample I
      sample_Q : out std_logic_vector(bits_adc-1 downto 0));  -- output sample Q
  end component;

  component multiplier
    port (
      clk  : in  std_logic;
      a    : in  std_logic_vector(7 downto 0);
      b    : in  std_logic_vector(7 downto 0);
      ce   : in  std_logic;
      sclr : in  std_logic;
      p    : out std_logic_vector(15 downto 0));
  end component;


  -------------------------------------------------------------------------------
  -- Signals Declaration
  -------------------------------------------------------------------------------
  signal signal_IN_1     : std_logic_vector(bits_adc-1 downto 0);
  signal signal_IN_2     : std_logic_vector(bits_adc-1 downto 0);
```

```
  signal signal_IN_track : std_logic_vector(bits_adc-1 downto 0);
  signal signal_I        : std_logic_vector(bits_adc-1 downto 0);
  signal signal_Q        : std_logic_vector(bits_adc-1 downto 0);
  signal result_I        : std_logic_vector(bits_iq-1 downto 0);
  signal result_Q        : std_logic_vector(bits_iq-1 downto 0);


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  signal_IN_track <= signal_IN_2;
  signal_IN_I     <= result_I;
  signal_IN_Q     <= result_Q;


  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  buffer_MEM_IQ_inst : buffer_MEM_IQ
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      sample_I => signal_I,
      sample_Q => signal_Q);

  multiplier_I_inst : multiplier
    port map (
      clk  => clk,
      a    => signal_IN_track,
      b    => signal_I,
      ce   => enable,
      sclr => rst,
      p    => result_I);

  multiplier_Q_inst : multiplier
    port map (
      clk  => clk,
      a    => signal_IN_track,
      b    => signal_Q,
      ce   => enable,
      sclr => rst,
      p    => result_Q);


  ---------------------------------------------------------------------------
  -- Delay Process to Synchronize DSSS Input Signal and I/Q Memory Signals
  ---------------------------------------------------------------------------
  delay : process (clk, rst) is
  begin  -- process delay

    if rst = '1' then                   -- asynchronous reset (active high)

      signal_IN_1  <= (others => '0');
      signal_IN_2  <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      signal_IN_1 <= signal_IN;
      signal_IN_2 <= signal_IN_1;

    end if;

  end process delay;

end rtl;
```

Código fuente A.24: buffer_MEM_IQ.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- BUFFER DEMODULATION I/Q
--------------------------------------------------------------------------------
entity buffer_MEM_IQ is

  port (
    rst      : in  std_logic;                                -- rst signal
    clk      : in  std_logic;                                -- clk signal
    enable   : in  std_logic;                                -- enable signal
    sample_I : out std_logic_vector(bits_adc-1 downto 0);    -- output sample I
    sample_Q : out std_logic_vector(bits_adc-1 downto 0));  -- output sample Q

end buffer_MEM_IQ;


architecture rtl of buffer_MEM_IQ is

  --------------------------------------------------------------------------------
  -- Components Declaration
  --------------------------------------------------------------------------------
  component memory_I
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples downto 0);
      data_out : out std_logic_vector(bits_adc-1 downto 0));
  end component;

  component memory_Q
    port (
      rst      : in  std_logic;
      clk      : in  std_logic;
      enable   : in  std_logic;
      r_w      : in  std_logic;
      address  : in  std_logic_vector(bits_samples downto 0);
      data_out : out std_logic_vector(bits_adc-1 downto 0));
  end component;


  --------------------------------------------------------------------------------
  -- Signals Declaration
  --------------------------------------------------------------------------------
  signal address : std_logic_vector(bits_samples downto 0);


begin  -- rtl

  --------------------------------------------------------------------------------
  -- Components Instantiation
  --------------------------------------------------------------------------------
  memory_I_inst : memory_I
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      r_w      => '0',
      address  => address,
      data_out => sample_I);
```

```
  memory_Q_inst : memory_Q
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      r_w      => '0',
      address  => address,
      data_out => sample_Q);


  ------------------------------------------------------------------------------
  -- I/Q Signals Reading Process
  ------------------------------------------------------------------------------
  mem_iq : process (clk, rst)
  begin  -- process mem_iq

    if rst = '1' then                   -- asynchronous reset (active high)

      address <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then

        if address < n_samples-1 then

          address <= address + 1;

        end if;

        if address = n_samples-1 then

          address <= (others => '0');

        end if;

      end if;

    end if;

  end process mem_iq;

end rtl;
```

Código fuente A.25: memory_I.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- I MEMORY (BRAM)
------------------------------------------------------------------------------
entity memory_I is

  port (
    rst      : in  std_logic;
    clk      : in  std_logic;
    enable   : in  std_logic;
    r_w      : in  std_logic;
    address  : in  std_logic_vector(bits_samples downto 0);
    data_out : out std_logic_vector(bits_adc-1 downto 0));

end memory_I;
```

```
architecture rtl of memory_I is

  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal r_w_i      : std_logic_vector(3 downto 0);
  signal address_i  : std_logic_vector(13 downto 0);
  signal data_in_i  : std_logic_vector(31 downto 0);
  signal data_out_i : std_logic_vector(31 downto 0);


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  r_w_i     <= (others => r_w);
  address_i <= "00" & address & "000";
  data_out  <= data_out_i(bits_adc-1 downto 0);


  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  -- RAMB16BWER: 16K-bit Data and 2K-bit Parity Configurable Synchronous Dual Port
  -- Block RAM with Optional Output Registers
  -- Spartan-6, Xilinx HDL Libraries Guide, version 12.1
  mem_I_inst : RAMB16BWER
    generic map (
      -- DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
      DATA_WIDTH_A        => 9,
      DATA_WIDTH_B        => 0,
      -- DOA_REG/DOB_REG: Optional output register (0 or 1)
      DOA_REG             => 1,
      DOB_REG             => 0,
      -- EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
      EN_RSTRAM_A         => true,
      EN_RSTRAM_B         => false,
      -- INIT_00 to INIT_3F: Initial memory contents.
      INIT_00             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_01             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_02             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_03             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_04             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_05             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_06             => X"80ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      INIT_07             => X"0000000080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f00",
      -- INIT_A/INIT_B: Initial values on output port
      INIT_A              => X"000000000",
      INIT_B              => X"000000000",
      -- INIT_FILE: Optional file used to specify initial RAM contents
      INIT_FILE           => "NONE",
      -- RSTTYPE: "SYNC" or "ASYNC"
      RSTTYPE             => "SYNC",
      -- RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
      RST_PRIORITY_A      => "SR",
      RST_PRIORITY_B      => "CE",
      -- SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY", "GENERATE_X_ONLY" or "NONE"
      SIM_COLLISION_CHECK => "ALL",
      -- SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
      SIM_DEVICE          => "SPARTAN6",
      -- SRVAL_A/SRVAL_B: Set/Reset value for RAM output
      SRVAL_A             => X"000000000",
      SRVAL_B             => X"000000000",
      -- WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
      WRITE_MODE_A        => "NO_CHANGE",
      WRITE_MODE_B        => "NO_CHANGE")
```

```
    port map (
      DOA    => data_out_i,             -- 32-bit A port data output
      DOB    => open,                   -- 32-bit B port data output
      DOPA   => open,                   -- 4-bit A port parity output
      DOPB   => open,                   -- 4-bit B port parity output
      ADDRA  => address_i,              -- 14-bit A port address input
      ADDRB  => "00" & x"000",          -- 14-bit B port address input
      CLKA   => clk,                    -- 1-bit A port clock input
      CLKB   => '0',                    -- 1-bit B port clock input
      DIA    => x"00000000",            -- 32-bit A port data input
      DIB    => x"00000000",            -- 32-bit B port data input
      DIPA   => "0000",                 -- 4-bit A port parity input
      DIPB   => "0000",                 -- 4-bit B port parity input
      ENA    => enable,                 -- 1-bit A port enable input
      ENB    => '0',                    -- 1-bit B port enable input
      REGCEA => '1',                    -- 1-bit A port register clock enable input
      REGCEB => '0',                    -- 1-bit B port register clock enable input
      RSTA   => rst,                    -- 1-bit A port register set/reset input
      RSTB   => '0',                    -- 1-bit B port register set/reset input
      WEA    => r_w_i,                  -- 4-bit Port A byte-wide write enable input
      WEB    => r_w_i);                 -- 4-bit Port B byte-wide write enable input
  -- End of RAMB16BWER_inst instantiation

end rtl;
```

Código fuente A.26: memory_Q.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library unisim;
use unisim.vcomponents.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- Q MEMORY (BRAM)
--------------------------------------------------------------------------------
entity memory_Q is

  port (
    rst      : in  std_logic;
    clk      : in  std_logic;
    enable   : in  std_logic;
    r_w      : in  std_logic;
    address  : in  std_logic_vector(bits_samples downto 0);
    data_out : out std_logic_vector(bits_adc-1 downto 0));

end memory_Q;


architecture rtl of memory_Q is

  --------------------------------------------------------------------------------
  -- Signals Declaration
  --------------------------------------------------------------------------------
  signal r_w_i      : std_logic_vector(3 downto 0);
  signal address_i  : std_logic_vector(13 downto 0);
  signal data_in_i  : std_logic_vector(31 downto 0);
  signal data_out_i : std_logic_vector(31 downto 0);


begin  -- rtl

  --------------------------------------------------------------------------------
  -- Signals Instantiation
  --------------------------------------------------------------------------------
  r_w_i     <= (others => r_w);
```

```
address_i <= "00" & address & "000";
data_out  <= data_out_i(bits_adc-1 downto 0);


--------------------------------------------------------------------------------
-- Components Instantiation
--------------------------------------------------------------------------------
-- RAMB16BWER: 16K-bit Data and 2K-bit Parity Configurable Synchronous Dual Port
-- Block RAM with Optional Output Registers
-- Spartan-6, Xilinx HDL Libraries Guide, version 12.1
mem_Q_inst : RAMB16BWER
  generic map (
    -- DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
    DATA_WIDTH_A        => 9,
    DATA_WIDTH_B        => 0,
    -- DOA_REG/DOB_REG: Optional output register (0 or 1)
    DOA_REG             => 1,
    DOB_REG             => 0,
    -- EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
    EN_RSTRAM_A         => true,
    EN_RSTRAM_B         => false,
    -- INIT_00 to INIT_3F: Initial memory contents.
    INIT_00             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_01             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_02             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_03             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_04             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_05             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_06             => X"0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    INIT_07             => X"0000000000080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f0080ff7f",
    -- INIT_A/INIT_B: Initial values on output port
    INIT_A              => X"000000000",
    INIT_B              => X"000000000",
    -- INIT_FILE: Optional file used to specify initial RAM contents
    INIT_FILE           => "NONE",
    -- RSTTYPE: "SYNC" or "ASYNC"
    RSTTYPE             => "SYNC",
    -- RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
    RST_PRIORITY_A      => "SR",
    RST_PRIORITY_B      => "CE",
    -- SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY", "GENERATE_X_ONLY" or "NONE"
    SIM_COLLISION_CHECK => "ALL",
    -- SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
    SIM_DEVICE          => "SPARTAN6",
    -- SRVAL_A/SRVAL_B: Set/Reset value for RAM output
    SRVAL_A             => X"000000000",
    SRVAL_B             => X"000000000",
    -- WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
    WRITE_MODE_A        => "NO_CHANGE",
    WRITE_MODE_B        => "NO_CHANGE")

  port map (
    DOA    => data_out_i,             -- 32-bit A port data output
    DOB    => open,                   -- 32-bit B port data output
    DOPA   => open,                   -- 4-bit A port parity output
    DOPB   => open,                   -- 4-bit B port parity output
    ADDRA  => address_i,              -- 14-bit A port address input
    ADDRB  => "00" & x"000",          -- 14-bit B port address input
    CLKA   => clk,                    -- 1-bit A port clock input
    CLKB   => '0',                    -- 1-bit B port clock input
    DIA    => x"00000000",            -- 32-bit A port data input
    DIB    => x"00000000",            -- 32-bit B port data input
    DIPA   => "0000",                 -- 4-bit A port parity input
    DIPB   => "0000",                 -- 4-bit B port parity input
    ENA    => enable,                 -- 1-bit A port enable input
    ENB    => '0',                    -- 1-bit B port enable input
    REGCEA => '1',                    -- 1-bit A port register clock enable input
    REGCEB => '0',                    -- 1-bit B port register clock enable input
    RSTA   => rst,                    -- 1-bit A port register set/reset input
    RSTB   => '0',                    -- 1-bit B port register set/reset input
```

```
      WEA    => r_w_i,                  -- 4-bit Port A byte-wide write enable input
      WEB    => r_w_i);                 -- 4-bit Port B byte-wide write enable input
  -- End of RAMB16BWER_inst instantiation

end rtl;
```

Código fuente A.27: filter_IQ.vhd

```
library ieee;
use ieee.std_logic_1164.all;

use work.constants_DDMS.all;

--------------------------------------------------------------------------------
-- FILTER I/Q
--------------------------------------------------------------------------------
entity filter_IQ is

  port (
    rst        : in  std_logic;                              -- reset signal
    clk        : in  std_logic;                              -- clock signal
    enable     : in  std_logic;                              -- enable signal
    data_IN_I  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data I
    data_IN_Q  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data Q
    data_OUT_I : out std_logic_vector(bits_iq-1 downto 0);   -- output data I
    data_OUT_Q : out std_logic_vector(bits_iq-1 downto 0));  -- output data Q

end entity filter_IQ;


architecture rtl of filter_IQ is

  --------------------------------------------------------------------------------
  -- Components Declaration
  --------------------------------------------------------------------------------
  component LPF_IQ is
    port (
      rst      : in  std_logic;                              -- reset signal
      clk      : in  std_logic;                              -- clock signal
      enable   : in  std_logic;                              -- enable signal
      data_IN  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data
      data_OUT : out std_logic_vector(bits_iq-1 downto 0));  -- output data
  end component;


begin  -- architecture rtl

  --------------------------------------------------------------------------------
  -- Components Instantiation
  --------------------------------------------------------------------------------
  LPF_IQ_I_inst : LPF_IQ
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      data_IN  => data_IN_I,
      data_OUT => data_OUT_I);

  LPF_IQ_Q_inst : LPF_IQ
    port map (
      rst      => rst,
      clk      => clk,
      enable   => enable,
      data_IN  => data_IN_Q,
      data_OUT => data_OUT_Q);

end architecture rtl;
```

Código fuente A.28: LPF_IQ.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- LPF I/Q
-------------------------------------------------------------------------------
entity LPF_IQ is

  port (
    rst      : in  std_logic;                              -- reset signal
    clk      : in  std_logic;                              -- clock signal
    enable   : in  std_logic;                              -- enable signal
    data_IN  : in  std_logic_vector(bits_iq-1 downto 0);   -- input data
    data_OUT : out std_logic_vector(bits_iq-1 downto 0));  -- output data

end entity LPF_IQ;


architecture rtl of LPF_IQ is

  -----------------------------------------------------------------------------
  -- Constants Declaration
  -----------------------------------------------------------------------------
  constant num_IQ_1 : std_logic_vector(bits_filter-1 downto 0) := X"0066";
  constant num_IQ_2 : std_logic_vector(bits_filter-1 downto 0) := X"0066";
  constant den_IQ_1 : std_logic_vector(bits_filter-1 downto 0) := X"7FFF";
  constant den_IQ_2 : std_logic_vector(bits_filter-1 downto 0) := X"80cd";


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal x_n   : std_logic_vector(bits_iq+bits_filter-1 downto 0);  -- input signal
  signal x_n_i : std_logic_vector(bits_iq-1 downto 0);              -- input signal
  signal x_n_1 : std_logic_vector(bits_iq+bits_filter-1 downto 0);  -- input signal delayed 1 sample
  signal y_n   : std_logic_vector(bits_iq+bits_filter-1 downto 0);  -- output signal
  signal y_n_1 : std_logic_vector(bits_iq+bits_filter-1 downto 0);  -- output signal delayed 1 sample


begin  -- architecture rtl

  -----------------------------------------------------------------------------
  -- Signals Instantiation
  -----------------------------------------------------------------------------
  data_OUT <= y_n(bits_iq+bits_filter-2 downto bits_filter-1);


  -----------------------------------------------------------------------------
  -- FIR I/Q Filter Process
  -----------------------------------------------------------------------------
  fir_IQ_filter : process (clk, rst) is
  begin  -- process fir_IQ:filter

    if rst = '1' then                   -- asynchronous reset (active high)

      x_n   <= (others => '0');
      x_n_i <= (others => '0');
      x_n_1 <= (others => '0');
      y_n   <= (others => '0');
      y_n_1 <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if enable = '1' then
```

```
        x_n_i <= data_IN;
        x_n   <= num_IQ_1*data_IN;
        x_n_1 <= num_IQ_2*x_n_i;
        y_n   <= x_n + x_n_1 - y_n_1;
        y_n_1 <= den_IQ_2*y_n(bits_iq+bits_filter-2 downto bits_filter-1);

      end if;

    end if;

  end process fir_IQ_filter;

end architecture rtl;
```

Código fuente A.29: phase_extractor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library UNISIM;
use UNISIM.vcomponents.all;

library XilinxCoreLib;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- PHASE EXTRACTOR
-------------------------------------------------------------------------------
entity phase_extractor is

  port (
    rst         : in  std_logic;        -- reset signal
    clk         : in  std_logic;        -- clock signal
    enable      : in  std_logic;        -- enable arctan
    signal_IN_I : in  std_logic_vector(bits_iq-1 downto 0);     -- input signal demodulated I
    signal_IN_Q : in  std_logic_vector(bits_iq-1 downto 0);     -- input signal demodulated Q
    phase       : out std_logic_vector(bits_phase-1 downto 0)); -- phase measured

end phase_extractor;


architecture rtl of phase_extractor is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component arc_tan is
    port (
      x_in      : in  std_logic_vector(15 downto 0);
      y_in      : in  std_logic_vector(15 downto 0);
      phase_out : out std_logic_vector(15 downto 0);
      rdy       : out std_logic;
      clk       : in  std_logic;
      ce        : in  std_logic;
      sclr      : in  std_logic);
  end component;


  -----------------------------------------------------------------------------
  -- Signals Declaration
  -----------------------------------------------------------------------------
  signal phase_ready : std_logic;
  signal phase_i     : std_logic_vector(bits_phase-1 downto 0);


begin  -- rtl
```

```
  ------------------------------------------------------------------------------
  -- Components Instantiation
  ------------------------------------------------------------------------------
  arc_tan_inst : arc_tan                -- arctang(Y/X) --> arctang(Q/I)
    port map (
      x_in      => signal_IN_I,
      y_in      => signal_IN_Q,
      phase_out => phase_i,
      rdy       => phase_ready,
      clk       => clk,
      ce        => enable,
      sclr      => rst);


  process (clk, rst)
  begin

    if rst = '1' then

      phase <= (others => '0');

    elsif clk'event and clk = '1' then

      phase <= phase_i;

    end if;

  end process;

end rtl;
```

Código fuente A.30: arc_tan.vhd

```
library ieee;
use ieee.std_logic_1164.all;
-- synthesis translate_off
library XilinxCoreLib;
-- synthesis translate_on

------------------------------------------------------------------------------
-- ARCTAN
------------------------------------------------------------------------------
entity arc_tan is

  port (
    x_in      : in  std_logic_vector(15 downto 0);
    y_in      : in  std_logic_vector(15 downto 0);
    phase_out : out std_logic_vector(15 downto 0);
    rdy       : out std_logic;
    clk       : in  std_logic;
    ce        : in  std_logic;
    sclr      : in  std_logic);

end arc_tan;


architecture arc_tan_a of arc_tan is

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  -- synthesis translate_off
  component wrapped_arc_tan
    port (
      x_in      : in  std_logic_vector(15 downto 0);
      y_in      : in  std_logic_vector(15 downto 0);
      phase_out : out std_logic_vector(15 downto 0);
      rdy       : out std_logic;
```

```
      clk       : in  std_logic;
      ce        : in  std_logic;
      sclr      : in  std_logic);
  end component;

  -- Configuration specification
  for all : wrapped_arc_tan use entity XilinxCoreLib.cordic_v4_0(behavioral)
    generic map(
      c_has_clk         => 1,
      c_has_x_out       => 0,
      c_has_y_in        => 1,
      c_reg_inputs      => 1,
      c_architecture    => 2,
      c_input_width     => 16,
      c_iterations      => 0,
      c_precision       => 0,
      c_has_rdy         => 1,
      c_has_sclr        => 1,
      c_scale_comp      => 0,
      c_has_nd          => 0,
      c_has_phase_in    => 0,
      c_has_rfd         => 0,
      c_cordic_function => 3,
      c_has_ce          => 1,
      c_round_mode      => 0,
      c_family          => "spartan6",
      c_has_y_out       => 0,
      c_data_format     => 0,
      c_reg_outputs     => 1,
      c_coarse_rotate   => 1,
      c_phase_format    => 0,
      c_has_x_in        => 1,
      c_has_phase_out   => 1,
      c_pipeline_mode   => -2,
      c_xdevicefamily   => "spartan6",
      c_output_width    => 16);
  -- synthesis translate_on


begin

  ------------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------------
  -- synthesis translate_off
  U0 : wrapped_arc_tan
    port map (
      x_in      => x_in,
      y_in      => y_in,
      phase_out => phase_out,
      rdy       => rdy,
      clk       => clk,
      ce        => ce,
      sclr      => sclr);
  -- synthesis translate_on

end arc_tan_a;
```

Código fuente A.31: diff_phase.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

------------------------------------------------------------------------------
-- DIFFERENTIAL PHASE
------------------------------------------------------------------------------
```

```
entity diff_phase is

  port (
    rst        : in  std_logic;         -- reset signal
    clk        : in  std_logic;         -- clock signal
    phase_i    : in  std_logic_vector(bits_phase-1 downto 0);  -- signal i phase
    phase_j    : in  std_logic_vector(bits_phase-1 downto 0);  -- signal j phase
    phase_diff : out std_logic_vector(bits_phase-1 downto 0)); -- differential phase

end diff_phase;


architecture rtl of diff_phase is

  ---------------------------------------------------------------------------
  -- Components Declaration
  ---------------------------------------------------------------------------
  component phase_correction is
    port (
      rst         : in  std_logic;      -- reset signal
      clk         : in  std_logic;      -- clock signal
      phase_ready : in  std_logic;      -- phase ready
      phase_in    : in  std_logic_vector(bits_phase-1 downto 0);   -- wrong phase
      phase_out   : out std_logic_vector(bits_phase-1 downto 0));  -- right phase
  end component;


  ---------------------------------------------------------------------------
  -- Signals Declaration
  ---------------------------------------------------------------------------
  signal phase_diff_i          : std_logic_vector(bits_phase downto 0);
  signal phase_diff_ii         : std_logic_vector(bits_phase downto 0);
  signal phase_diff_plus_2_pi  : std_logic_vector(bits_phase downto 0);
  signal phase_diff_minus_2_pi : std_logic_vector(bits_phase downto 0);
  signal phase_diff_current    : std_logic_vector(bits_phase downto 0);


begin  -- architecture rtl

  ---------------------------------------------------------------------------
  -- Components Instantiation
  ---------------------------------------------------------------------------
  phase_correction_inst : phase_correction
    port map (
      rst         => rst,
      clk         => clk,
      phase_ready => '1',
      phase_in    => phase_diff_current(bits_phase-1 downto 0),
      phase_out   => phase_diff);


  process (clk, rst) is
  begin  -- process

    if rst = '1' then                   -- asynchronous reset (active high)

      phase_diff_i          <= (others => '0');
      phase_diff_ii             <= (others => '0');
      phase_diff_plus_2_pi  <= (others => '0');
      phase_diff_minus_2_pi <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      -- 1 latency cycle. Sign Extension To Avoid Overflow
      phase_diff_i          <= (phase_i(bits_phase-1) & phase_i) - (phase_j(bits_phase-1) & phase_j);
      -- 2 latency cycles
      phase_diff_ii                <= phase_diff_i;
      phase_diff_plus_2_pi  <= phase_diff_i + pi_2_bin;
      phase_diff_minus_2_pi <= phase_diff_i - pi_2_bin;
```

```
    end if;

  end process;


  -------------------------------------------------------------------------------
  -- Differential Phase Correction Process
  -------------------------------------------------------------------------------
  correction : process (clk, rst)
  begin  -- correction

    if rst = '1' then                   -- asynchronous reset (active high)

      phase_diff_current <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if phase_diff_ii < -pi_bin then

        phase_diff_current <= phase_diff_plus_2_pi;

      elsif phase_diff_ii > pi_bin then

        phase_diff_current <= phase_diff_minus_2_pi;

      else

        phase_diff_current <= phase_diff_ii;

      end if;

    end if;

  end process correction;

end architecture rtl;
```

Código fuente A.32: phase_correction.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- PHASE CORRECTION
-------------------------------------------------------------------------------
entity phase_correction is

  port (
    rst         : in  std_logic;        -- reset signal
    clk         : in  std_logic;        -- clock signal
    phase_ready : in  std_logic;        -- phase ready
    phase_in    : in  std_logic_vector(bits_phase-1 downto 0);   -- wrong phase
    phase_out   : out std_logic_vector(bits_phase-1 downto 0));  -- right phase

end phase_correction;


architecture rtl of phase_correction is

  -------------------------------------------------------------------------------
  -- Signals Declaration
  -------------------------------------------------------------------------------
  signal previous_phase           : std_logic_vector(bits_phase downto 0);
  signal current_phase            : std_logic_vector(bits_phase downto 0);
  signal current_phase_plus_pi    : std_logic_vector(bits_phase downto 0);
  signal current_phase_minus_pi   : std_logic_vector(bits_phase downto 0);
```

```
  signal current_phase_i          : std_logic_vector(bits_phase downto 0);
  signal current_phase_plus_2_pi  : std_logic_vector(bits_phase downto 0);
  signal current_phase_minus_2_pi : std_logic_vector(bits_phase downto 0);
  signal phase_difference         : std_logic_vector(bits_phase downto 0);
  signal phase_out_i              : std_logic_vector(bits_phase downto 0);


begin  -- rtl

  ---------------------------------------------------------------------------
  -- Signals Instantiation
  ---------------------------------------------------------------------------
  phase_out <= phase_out_i(bits_phase-1 downto 0);


  process (clk, rst)
  begin

    if rst = '1' then

      previous_phase         <= (others => '0');
      current_phase          <= (others => '0');
      current_phase_plus_pi  <= (others => '0');
      current_phase_minus_pi <= (others => '0');

      current_phase_i          <= (others => '0');
      current_phase_plus_2_pi  <= (others => '0');
      current_phase_minus_2_pi <= (others => '0');
      phase_difference         <= (others => '0');

    elsif clk'event and clk = '1' then

      -- 1 latency cycle. Sign Extension To Avoid Overflow
      previous_phase         <= phase_out_i;
      current_phase          <= (phase_in(bits_phase-1) & phase_in);
      current_phase_plus_pi  <= (phase_in(bits_phase-1) & phase_in) + pi_bin;
      current_phase_minus_pi <= (phase_in(bits_phase-1) & phase_in) - pi_bin;
      -- 2 latency cycles
      current_phase_i          <= current_phase;
      current_phase_plus_2_pi  <= current_phase_plus_pi  + pi_bin;
      current_phase_minus_2_pi <= current_phase_minus_pi - pi_bin;
      phase_difference         <= current_phase - previous_phase;

    end if;

  end process;


  ---------------------------------------------------------------------------
  -- Current Differential Phase Correction Process
  ---------------------------------------------------------------------------
  correction : process (clk, rst)
  begin  -- correction

    if rst = '1' then                   -- asynchronous reset (active high)

      phase_out_i <= (others => '0');

    elsif clk'event and clk = '1' then  -- rising clock edge

      if phase_difference > threshold_2_pi_POS then

        phase_out_i <= current_phase_minus_2_pi;

      elsif phase_difference < threshold_2_pi_NEG then

        phase_out_i <= current_phase_plus_2_pi;

      else
```

```
        phase_out_i <= current_phase_i;

      end if;

    end if;

  end process correction;

end rtl;
```

Código fuente A.33: phase2distance.vhd

```
library ieee;
use ieee.std_logic_1164.all;

library UNISIM;
use UNISIM.vcomponents.all;

library XilinxCoreLib;

use work.constants_DDMS.all;

-------------------------------------------------------------------------------
-- PHASE TO DISTANCE CONVERSOR
-------------------------------------------------------------------------------
entity phase2distance is

  port (
    rst      : in  std_logic;           -- reset signal
    clk      : in  std_logic;           -- clk signal
    phase    : in  std_logic_vector(bits_phase-1 downto 0);  -- phase estimation
    distance : out std_logic_vector(bits_dist-1 downto 0));  -- distance estimation

end entity phase2distance;


architecture rtl of phase2distance is

  -----------------------------------------------------------------------------
  -- Components Declaration
  -----------------------------------------------------------------------------
  component multiplier_p2d is
    port (
      clk  : in  std_logic;
      a    : in  std_logic_vector(15 downto 0);
      b    : in  std_logic_vector(19 downto 0);
      ce   : in  std_logic;
      sclr : in  std_logic;
      p    : out std_logic_vector(35 downto 0));
  end component;


begin  -- architecture rtl

  -----------------------------------------------------------------------------
  -- Components Instantiation
  -----------------------------------------------------------------------------
  multiplier_p2d_inst : multiplier_p2d
    port map (
      clk  => clk,
      a    => phase,
      b    => sensitivity_dp,
      ce   => '1',
      sclr => rst,
      p    => distance);

end architecture rtl;
```

Código fuente A.34: multiplier_p2d.vhd

```
library ieee;
use ieee.std_logic_1164.all;
-- synthesis translate_off
library XilinxCoreLib;
-- synthesis translate_on

-------------------------------------------------------------------------------
-- PHASE TO DISTANCE MULTIPLIER
-------------------------------------------------------------------------------
entity multiplier_p2d is

  port (
    clk  : in  std_logic;
    a    : in  std_logic_vector(15 downto 0);
    b    : in  std_logic_vector(19 downto 0);
    ce   : in  std_logic;
    sclr : in  std_logic;
    p    : out std_logic_vector(35 downto 0));

end multiplier_p2d;


architecture multiplier_p2d_a of multiplier_p2d is

  -------------------------------------------------------------------------------
  -- Components Declaration
  -------------------------------------------------------------------------------
  -- synthesis translate_off
  component wrapped_multiplier_p2d
    port (
      clk  : in  std_logic;
      a    : in  std_logic_vector(15 downto 0);
      b    : in  std_logic_vector(19 downto 0);
      ce   : in  std_logic;
      sclr : in  std_logic;
      p    : out std_logic_vector(35 downto 0));
  end component;

  -- Configuration specification
  for all : wrapped_multiplier_p2d use entity XilinxCoreLib.mult_gen_v11_2(behavioral)
    generic map(
      c_a_width           => 16,
      c_b_type            => 1,
      c_ce_overrides_sclr => 0,
      c_has_sclr          => 1,
      c_round_pt          => 0,
      c_model_type        => 0,
      c_out_high          => 35,
      c_verbosity         => 0,
      c_mult_type         => 1,
      c_ccm_imp           => 0,
      c_latency           => 5,
      c_has_ce            => 1,
      c_has_zero_detect   => 0,
      c_round_output      => 0,
      c_optimize_goal     => 1,
      c_xdevicefamily     => "spartan6",
      c_a_type            => 0,
      c_out_low           => 0,
      c_b_width           => 20,
      c_b_value           => "10000001");
  -- synthesis translate_on


begin
```

```
  ------------------------------------------------------------------------
  -- Components Declaration
  ------------------------------------------------------------------------
  -- synthesis translate_off
  U0 : wrapped_multiplier_p2d
    port map (
      clk  => clk,
      a    => a,
      b    => b,
      ce   => ce,
      sclr => sclr,
      p    => p);
  -- synthesis translate_on

end multiplier_p2d_a;
```

Código fuente A.35: constants_DDMS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.functions_DDMS.all;

------------------------------------------------------------------------
-- CONSTANTS PACKAGE FOR DDMS
------------------------------------------------------------------------
package constants_DDMS is

  constant Tclk               : time;              -- signal clock period
  constant fm                 : integer;           -- modulation frequency
  constant fs                 : integer;           -- sampling frequency
  constant bits_adc           : integer;           -- number of ADC bits
  constant bits_prn           : integer;           -- number of PRN code bits
  constant muestras_chip      : integer;           -- PRN code samples/chip
  constant epochs             : integer;           -- PRN code epochs
  constant n_samples          : integer;           -- total processing samples
  constant bits_samples       : integer;           -- bits to represent n samples
  constant bits_corr          : integer;           -- bits to represent correlation values
  constant bits_filter        : integer;           -- bits to quantize filter coefficients
  constant bits_iq            : integer;           -- bits to quantize I/Q demodulation signals
  constant bits_phase         : integer;           -- bits to represent phase values
  constant bits_dist          : integer;           -- bits to represent distance values
  constant sensitivity_dp     : std_logic_vector;  -- distance/phase sensitivity [m/º] value in Q = [20]
  constant pi_bin             : std_logic_vector;  -- 1*pi value in Q = [17,0] format
  constant pi_2_bin           : std_logic_vector;  -- 2*pi value in Q = [17,0] format
  constant threshold_pi_POS   : std_logic_vector;  -- +1*pi threshold in Q = [17,0] format
  constant threshold_pi_NEG   : std_logic_vector;  -- -1*pi threshold in Q = [17,0] format
  constant threshold_2_pi_POS : std_logic_vector;  -- +2*pi threshold in Q = [17,0] format
  constant threshold_2_pi_NEG : std_logic_vector;  -- -2*pi threshold in Q = [17,0] format
  constant threshold_EL       : std_logic_vector;  -- early - late threshold

end constants_DDMS;


package body constants_DDMS is

  constant Tclk               : time     := 10.0 ns;
  constant fm                 : integer  := 25_000_000;
  constant fs                 : integer  := 100_000_000;
  constant bits_adc           : integer  := 8;
  constant bits_prn           : integer  := 6;
  constant muestras_chip      : integer  := 4;
  constant epochs             : integer  := 1;
  constant n_samples          : integer  := (2**bits_prn - 1)*muestras_chip*epochs;
  constant bits_samples       : integer  := log2ceil(n_samples);
  constant bits_corr          : integer  := 2*bits_adc + bits_samples;
  constant bits_filter        : integer  := 2*bits_adc;
  constant bits_iq            : integer  := 2*bits_adc;
  constant bits_phase         : integer  := 16;
  constant bits_dist          : integer  := 36;
```

```
  constant sensitivity_dp     : std_logic_vector(bits_dist-bits_phase-1 downto 0) := X"7A3B2";
  constant pi_bin             : std_logic_vector(bits_phase downto 0)             := '0' & X"6487";
  constant pi_2_bin           : std_logic_vector(bits_phase downto 0)             := '0' & X"C90F";
  constant threshold_pi_POS   : std_logic_vector(bits_phase downto 0)             := '0' & X"7E95";
  constant threshold_pi_NEG   : std_logic_vector(bits_phase downto 0)             := '1' & X"816A";
  constant threshold_2_pi_POS : std_logic_vector(bits_phase downto 0)             := '0' & X"D31C";
  constant threshold_2_pi_NEG : std_logic_vector(bits_phase downto 0)             := '1' & X"2CE3";
  constant threshold_EL       : std_logic_vector(bits_corr-1 downto 0)            := X"07A120";

end constants_DDMS;
```

Código fuente A.36: functions_DDMS.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--------------------------------------------------------------------------------
-- FUNCTIONS PACKAGE FOR DDMS
--------------------------------------------------------------------------------
package functions_DDMS is

  function log2ceil (n : integer) return integer;

end functions_DDMS;


package body functions_DDMS is

  ------------------------------------------------------------------------------
  -- Function to Obtain the Ceil Base 2 Logarithm
  ------------------------------------------------------------------------------
  function log2ceil (n : integer) return integer is

    variable m, p : integer;

  begin

    m := 0;
    p := 1;

    for i in 0 to n loop

      if p < n then

        m := m + 1;
        p := p * 2;

      end if;

    end loop;

    return m;

  end log2ceil;

end functions_DDMS;
```

Apéndice B

Herramientas y recursos

Las herramientas necesarias para la elaboración de este trabajo han sido:

- PC compatible
- Sistema operativo Windows 7
- Entorno de desarrollo Emacs 24.1
- Entorno de síntesis Xilinx ISE Design Suite 12.1
- Entorno de simulación Modelsim 6.5d
- Lenguaje de procesamiento matemático MATLAB R2012a
- Procesador de textos LaTeX
- Procesador de imágenes Microsoft Office Visio 2010
- Procesador de imágenes Inkscape
- Tarjeta de evaluación SP605